Universidad Complutense de Madrid

Facultad de Informática

Grado en Ingeniería Informática

TRABAJO DE FIN DE GRADO

DETECCIÓN DE FRUTAS EN ÁRBOLES

FRUIT DETECTION ON TREES

Departamento de Arquitectura de Computadores y Automática

Alejandro Torres Alonso

Director:

Carlos García Sánchez

Enero 2020

Curso académico 2019-2020

Índice

Resumen

La tarea que ocupa este proyecto es la conocida como detección de objetos en imágenes, que se encuentra hoy en pleno apogeo gracias a la aparición de arquitecturas de redes neuronales que hacen uso de la disciplina conocida como Deep Learning. La detección de objetos es una de las técnicas de Computer Vision que, a su vez, es un área de la subdisciplina de la Informática conocida como Inteligencia Artificial, la cual está cada día presente en la tecnología, gracias al impulso de las grandes multinacionales tecnológicas, que han visto en esta un gran nicho de mercado que puede suponer una auténtica revolución en tan distintos áreas como la agricultura, la estadística, la conducción automática, la videovigilancia o el reconocimiento facial.

Los objetos que se buscan reconocer en este proyecto son frutos en árboles que se encuentran en distintas fases de maduración. En concreto, se ha dispuesto de un conjunto de imágenes de mandarinos con sus frutos en distintos colores (verde y naranja) en función de la mentada maduración. Para esta labor se ha hecho empleo de distintas arquitecturas de Deep Learning, y de una api de Tensorflow en busca de llevar a cabo la técnica conocida como transferencia de aprendizaje, que consiste en reentrenar modelos ya preentrenados con el fin de reducir costes de tiempo y mejorar los resultados. Con esto se busca poder detectar y señalar mediante cuadros delimitadores, con la máxima precisión posible, todas las mandarinas posibles situadas en estos árboles.

Para esta labor se creará un dataset propio, haciendo uso de técnicas de anotación y, posteriormente, aumento de datos, para poder así realizar los distintos reentrenamientos cuyos resultados se analizarán apoyándose en las métricas del desafío de evaluación conocido como COCO.

Además, se evaluará la velocidad de inferencia de los modelos de detección empleados probando distintas unidades de procesamiento, como son las CPUs, las GPUs y el acelerador USB de Intel conocido como VPU Intel® Movidius™ Myriad™ X, haciendo empleo de la herramienta OpenVINO de Intel.

Palabras clave

Aprendizaje automático, visión por computador, redes neuronales convolucionales, transferencia de aprendizaje, IA en el borde, Tensorflow, conjunto de imágenes anotadas, detección de objetos, desafío/competición de evaluación, precisión media.

Abstract

The task of this project is known as object detection in images, which is now in full swing thanks to the emergence of neural network architectures that make use of the discipline known as Deep Learning. Object detection is one of the Computer Vision techniques which, also, is an area of the Computer Science subdiscipline known as Artificial Intelligence, which is present every day in technology, thanks to the support of large technology multinationals , who have seen in this area a great niche market that can be a real revolution in such different sectors as agriculture, stadistics, self-driving car, video surveillance or facial recognition.

The objects that are required to be recognized in this project are fruits in trees in different stages of maturation. Specifically, a set of images of tangerine trees with their fruits in different colors (green and orange) depending on maduration. For this work, it has been used different Deep Learning architectures, and a Tensorflow api looking for using the technique known as transfer learning, which consists of retraining pre-trained models in order to reduce time and improve the results. With this it is seek to be able to detect and indicate by bounding boxes, with the maximum precision, all the possible clementines located in these trees.

In this task, a custom dataset will be created using annotation techniques and, later, data augmentation, to be able to perform the different retraining and their results will be analyzed based on the metrics of the evaluation challenge known as COCO.

In addition, the inference speed of the detection models used will be evaluated by testing different processing units, such as CPUs, GPUs and the Intel USB accelerator known as Intel® Movidius ™ Myriad ™ X VPU, using the OpenVINO toolkit from Intel.

Key words

Machine learning, computer vision, convolutional neural networks, transfer learning, edge AI, Tensorflow, dataset, object detection, evaluation challenge, average precision.

1. Introducción

La adquisición de tecnologías enmarcadas en el área de la inteligencia artificial en la agricultura ha supuesto un cambio de paradigma a la hora de gestionar cultivos. Siendo este un sector de vital importancia de cara a un futuro que estará marcado por la superpoblación y la necesidad de buscar medios para poder alimentarla [1].

La introducción de esta tecnología ha traido consigo mejoras para este sector tales como la predicción y clasificación empleadas por los conocidos como invernaderos inteligentes [2] o la predicción metereológica en busca de mejorar el rendimiento de los cultivos con el mínimo coste [3]. También es interesante el uso de sensores que ayuden a los agricultures a tomar medidas en función de los datos que estos les brinden, o el empleo de drones que hagan uso de cámaras y sensores para controlar mejor grandes terrenos agrícolas [4].

Este proyecto, el cual ha sido sugerido por una empresa IT del ámbito del Smart Farming, se encuentra enfocado en el control de cultivos para, haciendo uso del subcampo de la Inteligencia Artificial conocido como Computer Vision, poder detectar el estado de maduración en el que se encuentran los frutos del árbol *Citrus reticulata*, más comunmente conocido como mandarino. Se hará empleo de distintas redes englobadas en el Deep Learning en busca de una alta precisión para la detección de los patrones de estos frutos, apoyándose en la transferencia de aprendizaje para evitar partir de cero en el entrenamiento de estas redes.

Para este labor nos apoyaremos en Tensorflow Object Detection API, realizando distintos entrenamientos con varias arquitecturas de redes neuronales convolucionales en busca de los mejores resultados en cuanto a precisión. A continuación, valiéndonos del kit de herramientas conocido como OpenVINO de Intel, evaluaremos la velocidad de ejecución de estas arquitecturas para distintos procesadores: la CPU y GPU integradas en el ordenador de trabajo, y el USB acelerador VPU Intel® Movidius™ Myriad™ X facilitado por el director del trabajo, que hace empleo de un módulo acelerador ASIC, que se encuentra englobado en el campo del Edge AI.

Estas tecnologías se encuentran reflejadas en la gráfica de 2019 del conocido como ciclo de sobreexpectación de Gartner [5], que sirve para conocer la situación en cuanto a desarrollo, aceptación y puesta en producción de las tecnologías empleadas. En concreto, se observan el Transfer Learning y el Edge AI en la fase de lanzamiento, lo que implica que las espectativas con estos campos se encuentran en ascenso.

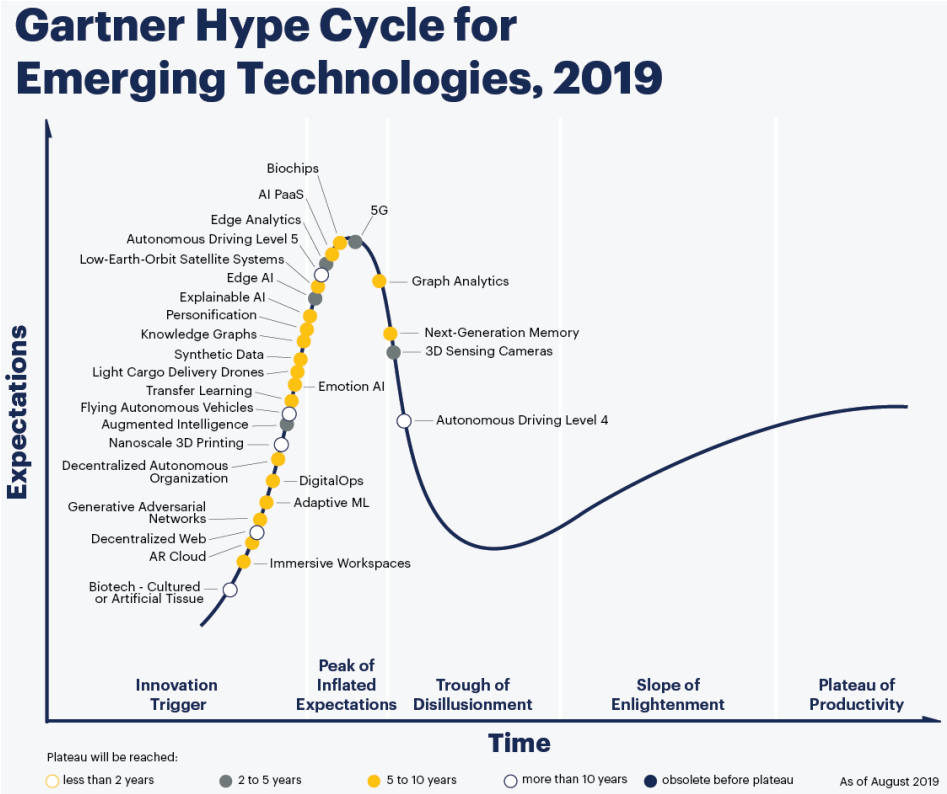

Imagen 1. Ciclo de Gartner para tecnologías emergentes de 2019

La memoria se compone de otros 6 capítulos, entre los que se encuentra esta misma introducción y la bibliografía. En los siguientes capítulos se realizará un recorrido a lo largo del estado del arte de la inteligencia artificial para continuar después con la metodología empleada con el objetivo de obtener los distintos resultados y conclusiones. Estos están ordenados como sigue:

- Estado del arte de la inteligencia artificial. En este extenso capítulo se pondrán las bases sobre las que se basa este campo de la informática, centrándose en las técnicas y arquitecturas del subcampo del computer vision y realizando un acercamiento al hardware específico empleado en esta área.
- Metodología. Este capítulo se divide en dos secciones diferenciadas, encontrándose la primera dedicada a las técnicas de tratamiento de los conjuntos de imágenes y de evaluación de los resultados, mientras que en la segunda se explican los pasos llevados en la puesta a punto y ejecución de la transferencia de aprendizaje.
- Resultados. Se realizará un análisis tanto del reentrenamiento llevado a cabo en la transferencia de aprendizaje como de los resultados obtenidos en estos tanto teórica como visualmente. Además se analizará la eficiencia de estos resultados al ejecutarlos en distintas arquitecturas.
- Conclusiones. Aquí se exponen las conclusiones a las que se ha llegado tras el análisis de los resultados, detallando también el alcance de este proyecto.

Los archivos más relevantes empleados en este trabajo, como el dataset customizado, los archivos de configuración, las gráficas de pérdida de entrenamiento y evaluación, inferencias de Tensorflow u OpenVINO, y ejecuciones con cuadernos de Jupyter se encuentran localizados en el siguiente repositorio GitHub.[1]

[1] https://github.com/alotorres/Proyecto

2. Introduction

The introduction of technologies related to the area of artificial intelligence in agriculture has meant a paradigm shift when it comes to managing crops. This is a sector of vital importance for a future that will be marked by overpopulation and the need to find ways to feed this [1].

The introduction of this technology has brought improvements for this sector such as the prediction and classification used by the known as smart greenhouses [2] or the weather forecast in search of improving crop yields with minimum cost [3]. It is also interesting to use sensors that help farmers take measures based on the data they provide, or using drones that use cameras and sensors to inprove the control of large agricultural lands [4].

This project, which has been suggested by an IT company specialized in the Smart Farming area, is focused on the control of crops, making use of the subfield of Artificial Intelligence known as Computer Vision, to detect maturation state of the fruits of *Citrus reticulata*, more commonly called tangerine tree. Different networks included in Deep Learning will be used in search of high precision for the detection of the patterns of these fruits, relying on the transfer learning to avoid starting from scratch in the training of these networks.

For this task we will rely on Tensorflow Object Detection API, performing different trainings with several convolutional neural network architectures in search of the best results in terms of accuracy. Next, using the toolkit known as Intel OpenVINO, we will evaluate the execution speed of these architectures for different processors: the CPU and GPU integrated in the work computer, and the USB VPU accelerator Intel® Movidius ™ Myriad ™ X provided by the director, which uses an ASIC accelerator module, and its included in the Edge AI field.

These technologies are reflected in the 2019 Gartner hype cycle chart [5], which serves to know the situation in terms of developtment, acceptance and implementation in production of the technologies used. Specifically, Transfer Learning and Edge AI are observed in the launch phase, which implies that expectations with these fields are on the rise.

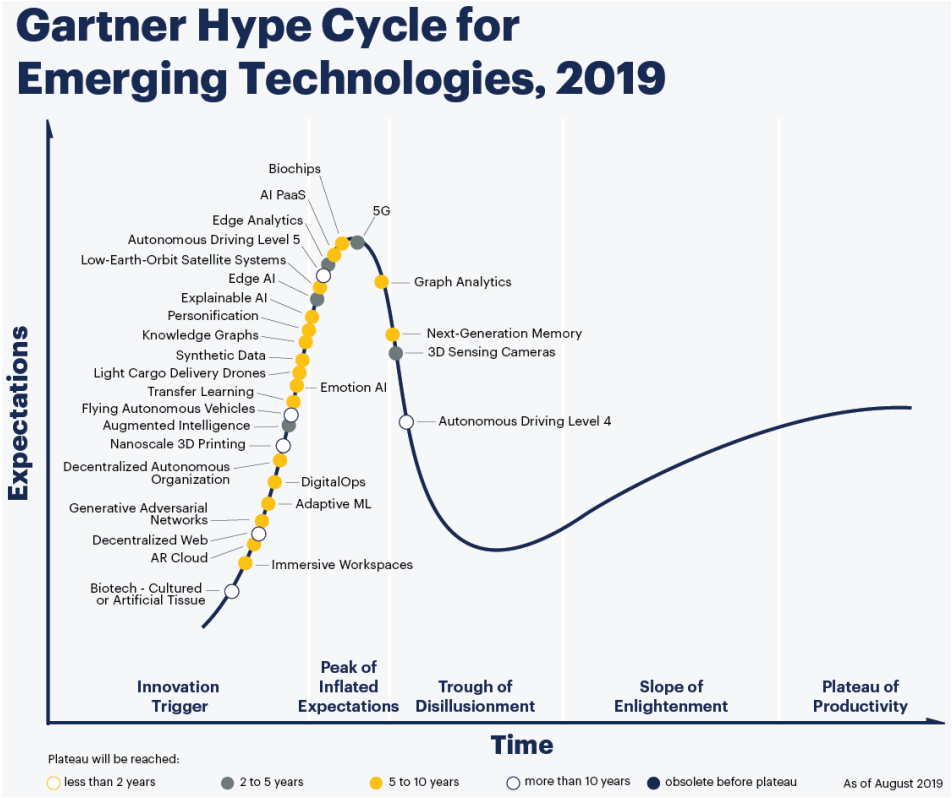

Imagen 2. Ciclo de Gartner para tecnologías emergentes de 2019

The report consists of 6 other chapters, among which is the same introduction and the bibliography. In the following chapters a tour will be made throughout the state of the art of artificial intelligence to continue later with the methodology used in order to obtain the different results and conclusions. These are ordered as follows:

- State of the art of artificial intelligence. This large chapter will lay the foundations on which this computing field is based, focusing on the techniques and architectures of the computer vision subfield and making an approach to the specific hardware used in this area.
- Methodology. This chapter is divided into two distinct sections, the first one being dedicated to the techniques of treatment of the image sets and the results evaluation, while the second explains the steps taken in the set-up and execution of the transfer learning.
- Results. It will make an analysis of the training carried out in the transfer learning and of the results obtained in these both theoretically and visually. In addition, the efficiency of these results will be analyzed by executing them in different architectures.
- Conclusions. Here are the conclusions reached after the analysis of the results, also detailing the scope of this project.

The most relevant files used in this project, such as custom dataset, configuration files, training and evaluation loss charts, Tensorflow or OpenVINO inferences, and executions with Jupyter notebooks are located in the following GitHub repository[2].

[2] https://github.com/alotorres/Proyecto

3. Estado del arte de la inteligencia artificial

En el estado del arte de este proyecto se trata el campo de la IA, partiendo de una introducción a esta y a los áreas en los que se subdivide, para luego enfocarse en los puntos teóricos que van a ser tratados a lo largo del proyecto, como el deep learning y las redes neuronales convolucionales, o las técnicas y arquitecturas empleadas en el subcampo de la visión por computador. También se realizará un repaso de los frameworks y hardware empleados en tareas de este campo.

3.1. Introducción a la IA

Antes de comenzar con los preparativos y puesta a punto de los experimentos relativos a este proyecto, será preciso poner en contexto las distintas ramas de la informática que van a ser tratadas.

Este proyecto se encuentra enmarcado en la subdisciplina del campo de la Informática conocida como inteligencia artificial (del inglés, artificial intelligence, AI). Esta área de la informática busca el estudio y la creación de programas o máquinas que puedan aprender en base a datos de entrada con el fin de resolver problemas, reproduciendo conductas inteligentes ligadas al ser humano.

Su nacimiento como concepto data del año 1950, cuando el británico Allan Turing publicó "Computing Machinery and Intelligence", introduciendo al público el conocido "Test de Turing", que se cuestionaba si las máquinas pueden pensar [6].

La inteligencia artificial, a su vez, se subdivide en distintas áreas de investigación, que suelen estar interconectadas entre ellas. Destacan el *procesamiento de lenguaje natural (PNL)*, los *sistemas expertos*, la *robótica*, la *visión por computador* y el *aprendizaje automático*. Este trabajo se centra en las dos últimas.

La *visión por computador* (del inglés, *computer vision*), es un campo que busca, mediante la implementación de técnicas específicas, dotar a los ordenadores de la habilidad de "ver" y "comprender" la información contenida dentro de fotogramas (imágenes), para lograr así la automatización de distintas tareas que los humanos podríamos realizar gracias a nuestro sistema visual [7].

Por su parte, *aprendizaje automático* (del inglés, *machine learning*) busca capacitar a los ordenadores de la habilidad de aprender por sí mismos. Puede decirse que esta área de investigación es una alternativa a la convencional forma en que la ingeniería diseña algoritmos en busca de soluciones, la cual se basaba en ejecutar una serie de reglas diseñadas manualmente sobre la entrada del programa. Con el aprendizaje automático, en cambio, se busca diseñar un

programa que, mediante el aprendizaje de patrones comunes para un número lo suficientemente grande de ejemplos de entrada que cumplan un comportamiento deseado, consiga localizar los mismos patrones en nuevos datos de entrada [8].

3.2. Modelos de aprendizaje automático

Dentro del aprendizaje automático existen diversas taxonomías de algoritmos, las cuales pueden ser clasificadas según su salida:

- En el *aprendizaje supervisado* se parte de unos datos de entrenamiento basados en pares de objetos etiquetados: la entrada y la salida deseada, existiendo una correspondencia entre estos. Se requiere de empleo de mano humana para este llevar a cabo el etiquetado de los elementos.
- El *aprendizaje no supervisado*, en cambio, no cuenta con datos etiquetados para la salida teniendo así que estar capacitado el sistema para poder etiquetar en base a los patrones de entrada que reconozca.
- Por otro lado, se encuentra el *aprendizaje semi-supervisado*, el cual se sitúa entre las dos taxonomías explicacadas anteriormente, al emplear tanto datos etiquetados como no etiquetados. Implica una mejora respecto a las otras taxonomías ya que, por un lado, mejora los resultados en cuanto a precisión obtenidos en el aprendizaje no supervisado al hacer empleo de datos etiquetados y, por el otro, su coste será inferior al del aprendizaje supervisado al no requerir de tanta mano humana cuando el conjunto de datos es muy grande, ya que, la cantidad de datos etiquetados será muy inferior a la de no etiquetados.
- A su vez, se encuentra el *aprendizaje reforzado*, que se basa en técnicas de retroalimentación, ensayo-error, con el fin de mejorar la respuesta del modelo.
- Otras taxonomías de algoritmos de aprendizaje automático serían la *transducción* o el *aprendizaje multitarea*.

3.3. Técnicas de aprendizaje automático

En los algoritmos de machine learning se emplean bastantes técnicas que buscan obtener información en función de los datos de entrada. Entre estas, destacan las conocidas como *redes neuronales artificiales* (ANN), que están basadas en las sistemas nerviosos de los organismos vivos. Estas tienen una estructura flexible, organizada en distintos niveles (capas) formados por una gran cantidad de elementos de procesamiento, conocidos como neuronas, que se encuentran interconectadas unas con otras, Estas neuronas, que son funciones matemáticas, tienen como output el producto de una función no lineal, llamada función de activación, por la suma ponderada

de sus entradas y pesos de estas. Las entradas se corresponden a los estímulos que rodean a la neurona, y la salida a la respuesta que emite la neurona ante estos estímulos, siendo esta salida la que recibe como entrada las neuronas de las capas posteriores [9].

Durante el proceso de entrenamiento de las redes neuronales se hace uso de la conocida como función de pérdida, la cual permite conocer cómo de buena es la red para una tarea específica. Cuanto mayor sea el valor de esta función, peor serán los resultados que obtengamos con la red, por tanto, se busca minimizar este valor. Al comenzar el entrenamiento se elegirán unos pesos al azar, que provocarán muy malos resultados y, durante el proceso de entrenamiento, estos pesos se irán ajustando al problema, minimizándose la pérdida.

Otra técnica importante es la *transferencia de aprendizaje* (del inglés, *transfer learning*), que permite que se pueda emplear conocimiento adquirido de un modelo ya entrenado para un conjunto enorme de datos con el fin de poder reentrenarlo con nuevos datos, lográndose unos mejores resultados cuando mayor correlación se guarde entre los nuevos datos y lo viejos. Esta técnica ha supuesto un gran avance en los últimos años, dado que permite realizar el entrenamiento de conjuntos de datos sin necesidad de hacerlo desde cero, consiguiendo así reducir enormemente la duración y el coste computacional de este. También permite reducir el tamaño del conjunto de datos gracias a la generalización del conjunto grande empleado en el entrenamiento inicial del modelo, suponiendo también un aumento en la precisión de la detección [10].

3.4. Deep learning

El *aprendizaje profundo* (del inglés, *deep learning*) es una disciplina que surge de las redes neuronales artificiales, y se compone del entrelazado de estas, aumentando el número de capas y su complejidad. Es una rama del aprendizaje automático cuyo objetivo es permitir el aprendizaje de representaciones de datos con distintos niveles de abstracción, empleando para esto distintas arquitecturas que se encuentran compuestas por una cantidad indeterminada de niveles de computación [11].

Como puede observarse en la imagen siguiente, en este modelo se emplean un número variable de niveles o capas, en los que, la abstracción y complejidad es creciente. Se hace uso del término profundo porque, pese que estos contengan una gran cantidad de capas, solo puede accederse a la primera y a la última, conociéndose las capas interiores como *"capas ocultas"* (del inglés, hidden layers).

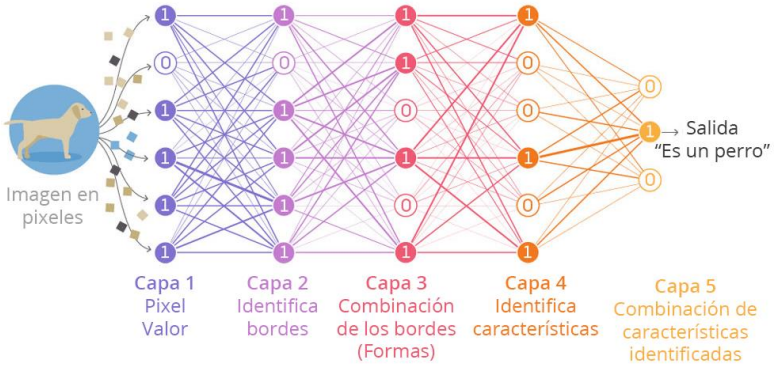

Imagen 3. Ejemplo del funcionamiento de deep learning haciendo uso de una red neuronal artificial. Obtenida en el siguiente blog[3].

Pese a que en la actualidad se encuentra en boga, la primera publicación de un algoritmo de deep learning data del año 1967, cuando Alexey Ivakhnenko y Valentin G. Lapa introdujeron el primer algoritmo de aprendizaje para perceptrones [12] multicapa, supervisados y profundos en su publicación *"Cybernetics and forecasting techniques"* [13]. Pero no ha sido práctica su implementación hasta que el tiempo de cálculo de los procesos, tanto en CPUs como en GPUs, no se ha visto reducido drásticamente, y hasta que no se ha dispuesto de masivas cantidades de datos de prueba para los entrenamientos. Es en este momento, coincidiendo con la publicación en 2006 del artículo "Deep Learning" de Geoffrey Hinton [11], cuando entran en juego grandes compañías tecnológicas, como IBM, Facebook, Amazon o Google, para darle un cariz comercial e impulsarlo enormemente hasta convertirse en lo que es hoy en día.

El empleo de esta disciplina ha supuesto una mejora considerable del estado del arte de técnicas tales como el reconocimiento de voz y la clasificación y detección de objetos visuales. También ha supuesto un progreso en campos como la biomedicina [14] o la genómica [15], e incluso se ha empleado para mejorar el cómputo de expresiones matemáticas [16].

Como se mencionó previamente, las distintas áreas de investigación de la inteligencia artificial están interrelacionadas. Es por esto por lo que el aprendizaje profundo, pese a ser una subdisciplina del aprendizaje automático, se encuentra ampliamente relacionado con la visión por computador.

3.5. Clasificación, detección y segmentación

Dentro del campo de la visión por computador se emplean distintas técnicas en función de la imagen que se busca analizar. Estas hacen empleo de conjuntos de imágenes anotadas, conocidos como datasets. A continuación, se realiza un repaso de las más destacadas:

[3] https://www.smartpanel.com/que-es-deep-learning/

La *clasificación* de imágenes es la técnica más extendida dentro de este campo. Busca clasificar imágenes en distintas categorías o clases, es decir, se cuenta con un dataset de entrada consistente en N imágenes etiquetadas con M clases diferentes. En este tipo de problemas la salida suele consistir en señalar para cada imagen qué porcentaje de cada clase se encuentra presente en esta.

Imagen 4. Ejemplo de clasificación, a una imagen se le asginan distintos atributos o clases, como por ejemplo un gato, un pero o una flor. Imagen obtenida en la siguiente noticia[4].

La *localización* de imágenes busca detectar la o las posiciones donde se encuentra una clase concreta en la imagen y señalarla mediante un cuadro delimitador (del inglés, bounding box). Por tanto, el input consistirá en un conjunto de N imágenes a las que se le tiene que añadir un archivo que señale en qué puntos se sitúan los objetos etiquetados, que pertenecerán a la misma clase.

Imagen 5. Ejemplo de localización, donde se señalan todos los bounding boxes de una clase concreta, en este caso la clase "flor".

Por otro lado, se encuentra la *detección* de objetos, técnica que busca unificar las dos anteriores, la clasificación y la localización, en busca de precisar la información relativa a la imagen al poder etiquetar distintos objetos de distintas clases. La entrada, al igual que en la localización, hace empleo de imágenes junto a anotaciones, aunque en las anotaciones en este caso guardan la información de los bounding boxes de cada una de las apariciones de cada una de las clases aparecidas en la imagen.

Imagen 6. Ejemplo de clasificación, donde para cada clase localizada en la imagen se señala un atributo.

[4] https://www.abc.es/sociedad/mascotas/abci-mas-barato-mantener-perro-o-gato-201910100240_noticia.html

La *segmentación semántica* [17] es un proceso que consiste en vincular a distintas clases cada uno de los píxeles de una imagen, buscando así comprender el papel de cada píxel en esta. Al igual que en detección de objetos, se detectan distintos objetos en una imagen, aunque la delimiticación de estos es por píxeles, ajustándose así mejor al objeto y siendo más precisa.

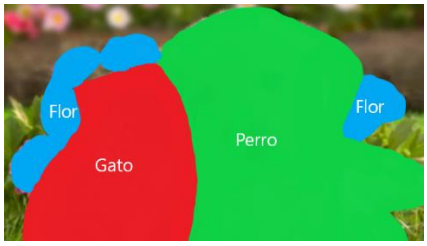

Imagen 7. Ejemplo de segmentación semántica

La *segmentación de instancia* va más allá que la segmentación semántica, permitiendo diferenciar entre distintos objetos de la misma o distintas clases superpuestos, haciendo mucho más precisa la detección.

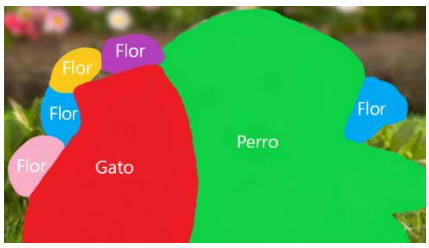

Imagen 8. Ejemplo de segmentación por instancia. A diferencia de la segmentación semántica, consigue hacer distinción de distintos objetos de una misma clase que se encuentran juntos.

Las aquitecturas que son empleadas para solucionar los problemas de visión por computador son explicadas en el punto siguiente.

3.6. Arquitecturas de Computer Vision

Una vez conocidas las distintas técnicas empleadas en el campo de la visión por computador, cabe realizar un recorrido a lo largo de las arquitecturas en los que estas se basan.

Una *red neuronal convolucional (CNN o ConvNet)* es un algoritmo de aprendizaje profundo capaz de reconocer distintos patrones de estímulo y diferenciar unos de otros, asignando pesos y sesgos a estos patrones, sin verse afectada por cambios o pequeñas distorsiones en estos [18]. Estas redes están inspiradas en la red neuronal artificial jerárquica de varias capas propuesta por Kunihiko Fukushima en 1980 [19], llamada Neocognitrón.

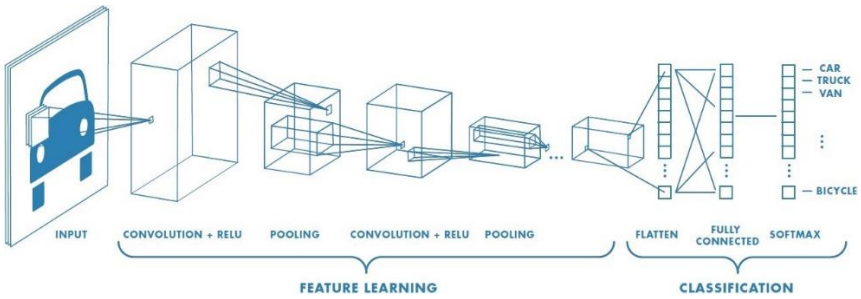

Imagen 9. Estructura de una red neuronal convolucional. Obtenida en el siguiente blog[5].

Estas redes se dividen en distintas fases. Para una entrada consistente en una imagen, se aplica en un comienzo una capa convolucional con el objetivo de extraer las características de bajo nivel. A los resultados de esta capa convolucional se les aplicará una función de activación conocida como ReLU. A continuación, se hace empleo de una capa conocida como pooling (agrupación), que reducirá el coste computacional requerido gracias a que disminuye el tamaño de las imágenes de forma que sea más fácil de procesar, siendo esto bastante útil a la hora de extraer las características dominantes. Existen distintas técnicas de agrupación, destacando max pooling y average pooling. Estas fases conformadas por capas convolucionales y capas de agrupación se irán repitiendo según cómo haya sido configurada la red. Cuanto más profundas sean, las capas convolucionales extraerán características de más alto nivel [18].

La última fase, es la capa completamente conectada (fully conected layer), la cual, tras finalizar las capas convolucionales, realiza un razonamiento de alto nivel de los resultados obtenidos.

En función de la distribución y tamaño de las capas convolucionales, de activación y de agrupación, y de la cantidad de parámetros empleados, se distinguen distintas arquitecturas de CNN, siendo las más conocidas AlexNet [20], VGGNet [21], ResNet [22] e Inception [23], las cuales han ido siendo publicadas a lo largo de los últimos años con la idea de conseguir buenos resultados en el desafío de clasificación de imágenes conocido como ImageNet [24].

Las redes CNN se emplean en técnicas de clasificación ya que consiguen que el procesamiento requerido de la imagen de entrada sea bastante menor que en otros algoritmos de clasificación, suponiendo un entrenamiento más fácil. Por otro lado, estas no son muy eficientes para la detección de objetos, ya que para poder aplicarlas se debería emplear un gran número de localizaciones a distintas escalas, suponiendo un enorme coste computacional. Es por esto por lo que surgen nuevas arquitecturas cuyo coste en tareas de detección es menor.

En 2013, Ross Girhsick propone las denominadas *redes neuronales convolucionales regionales (R-CNNs)*. Con estas se consigue obtener buenos resultados en las tareas que requieren de la detección de objetos. Esta arquitectura se divide en cuatro etapas, coincidiendo la primera con la

[5] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

imagen de entrada, y haciendo empleo en la segunda de un método denominado búsqueda selectiva (del inglés, selective search) [25], el cual clasifica las imágenes en un cantidad de hasta 2000 regiones, siendo estas candidatas a ser un objeto. Para cada una de estas propuestas se calcula un vector de características de 4096 dimensiones, el cual será la salida. En la tercera etapa es donde entra en juego un modelo CNN preentrenado, el cual extraerá los mapas de características que en la última etapa se pasarán por un SVM (Support Vector Machine), que clasificará la presencia del objeto en esta región candidata [26].

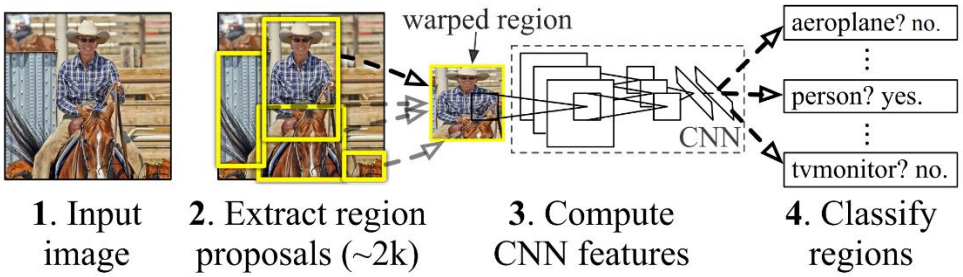

Imagen 10. Estructura de una R-CNN. Obtenida en el siguiente artículo[6].

Estas redes, pese a que introducen una mejora respecto de las CNNs convencionales, siguen requiriendo de una gran cantidad de tiempo de procesamiento, tomando alrededor de 47 segundos por cada imagen de prueba.

Por otro lado, se encuentra el algoritmo de *agrupación de pirámides espaciales (SPP-Net)*, publicado en 2015, el cual hace una agrupación multi-escala de las características de la imagen en búsqueda de la mejora del rendimineto de la red a la hora de clasificar y detectar, y también hace uso de inputs de imágenes de tamaño completamente arbitrario en el entrenamiento de la red neuronal convolucional, suponiendo una mejora respecto de las CNN básicas, las cuales hacían un redimensionado en las imágenes que podía provocar una distorsión en estas [27].

En 2015, el mismo Ross Girhsick introduce *Fast R-CNN* en busca de solventar los problemas que arrastraba R-CNN, lo que no solo implica una ejecución más rápida sino que la precisión media es más alta. Esto se debe a que emplea la imagen completa con la CNN, obteniéndose las características de la última capa de la convolución. En esta última capa se tienen dos outputs, uno denominado softmax, donde se decide a qué clase pertenece el objeto, y con la otra se obtienen las coordenadas del bounding box para cada clase [28].

Este cambio de estructura consigue que no se tengan que alimentar todas las regiones mentadas antes, que podían llegar a ser 2000, a la vez. Pero, como desventaja, esta red sigue empleando el método de la búsqueda selectiva, el cual es un proceso lento que afecta al rendimiento de la red.

[6] http://islab.ulsan.ac.kr/files/announcement/513/rcnn_pami.pdf

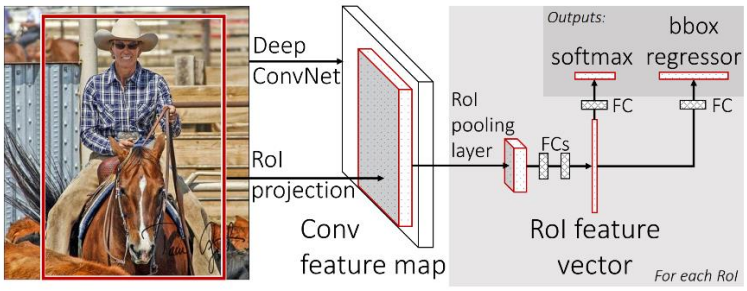

Imagen 11. Estructura de Fast R-CNN. Obtenida en el artículo[7].

En busca de conseguir eliminar del proceso los algoritmos de propuesta de región basados en CPU (como la búsqueda selectiva), se presenta en el mismo año 2015 el artículo que asienta las bases de la que es la arquitectura de la familia R-CNN más ampliamente empleada: *Faster R-CNN*. Esta sustituye estos algoritmos de propuesta de región por otra red convolucional conocida como red de propuestas de región (RPN), a la que acompaña una Fast R-CNN, que hará las veces de detector de propuestas de redes, esto es, el módulo RPN indicará al módulo Fast R-CNN dónde tiene que poner antención. Con esto se consigue unificar el sistema entero como sola red unificada de detección de objetos [29].

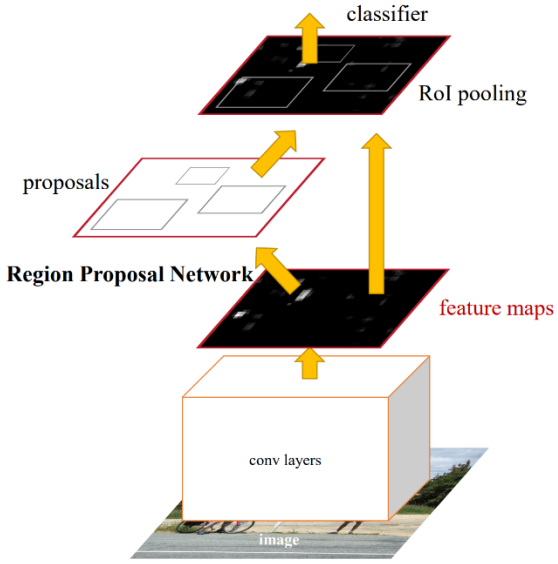

Imagen 12. Estructura de Faster R-CNN. Obtenida en el artículo[8].

Estas arquitecturas de detección de objetos basadas en propuestas de regiones tienen una tasa de aciertos muy alta, contando con tiempos de ejecución bastante correctos, por no mencionar que cuanto más actual es la versión, más rápidos y precisos son. El mayor problema que tienen estas surge cuando se quieren emplear en procesamiento de imágenes para detección de objetos en tiempo real, donde se tiene que descomponer el vídeo en frames, detectar para cada frame los objetos buscados y dibujarlos, para después montar el vídeo de nuevo. Realizar esto en tiempo

[7] https://arxiv.org/pdf/1504.08083.pdf
[8] https://arxiv.org/pdf/1506.01497.pdf

real con una cantidad de FPS que hagan que la calidad del vídeo no se vea menguada, requiere de algoritmos mucho más rápidos. Es por ello por lo que se emplean las conocidas como arquitecturas de una fase, también conocidas como single-shot, es decir, la imagen solo tiene que ser leída una vez. Las arquitecturas más conocidas de este tipo son YOLO y SSD.

YOLO (You Only Look Once) es presentada en 2015 por investigadores de la Universitiy of Washington, y hace empleo de una red neuronal para dividir una imagen en regiones sobre la que se aplica una CNN con el fin de predecir cada bounding box y las probabilidades de cada región [30]. YOLO ha recibido actualizaciones con los años, con YOLOv2 [31] y su última versión, YOLOv3 [32], que han ido suponiendo una mejora tanto en precisión como en velocidad respecto a la primera versión lanzada.

Por otro lado, se encuentra la arquitectura *SDD (Single Shot Multibox Detector)*, presentada a finales de 2016. Esta es significativamente más rápida y precisa que otras arquitecturas single-shot anteriores, como YOLO. Al hacer empleo de capas convoluciones de distintos tamaños, mejora la detección de objetos de tamaños variables, que en YOLO era bastante imprecisa, pero sin llegar al nivel de Faster R-CNN. También se eliminan las capas completamente conectadas (fully connected) intermedias, de las que sí dispone YOLO, que suponen un aumento del coste computacional [33].

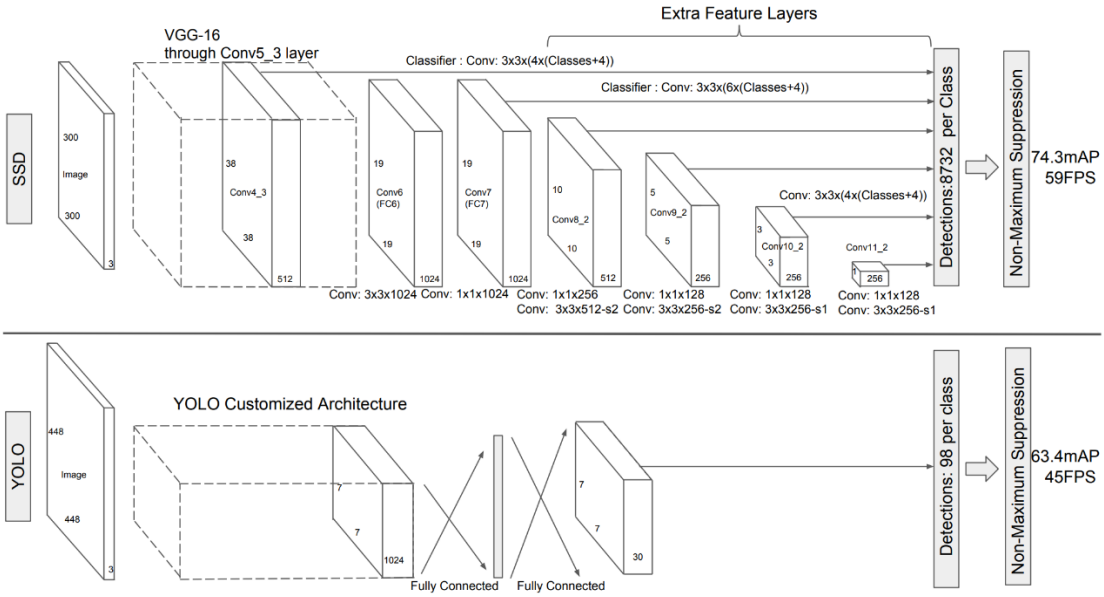

Imagen 13. Estructura de SSD (arriba) y de YOLO (abajo) . Obtenida en el artículo[9].

Hay que destacar también la arquitectura empleada para problemas de sementación de instancia, conocida como *Mask R-CNN*. Presentada en 2017, extiende Faster R-CNN añadiendo una rama para predecir máscaras sementadas en cada región de interés, siendo esta paralela a la empleada

[9] https://arxiv.org/pdf/1512.02325.pdf

para la detección de los bounding boxes. Esto es, si Faster R-CNN tenía como output las clases y los cuadros delimitados, Mask añade a estos una máscara binaria haciendo uso de una operación RoIAlign para cada region de interés (RoI). Con esta técnica se consigue superar a Faster R-CNN en frames por segundo, llegando así a 5 fps [34].

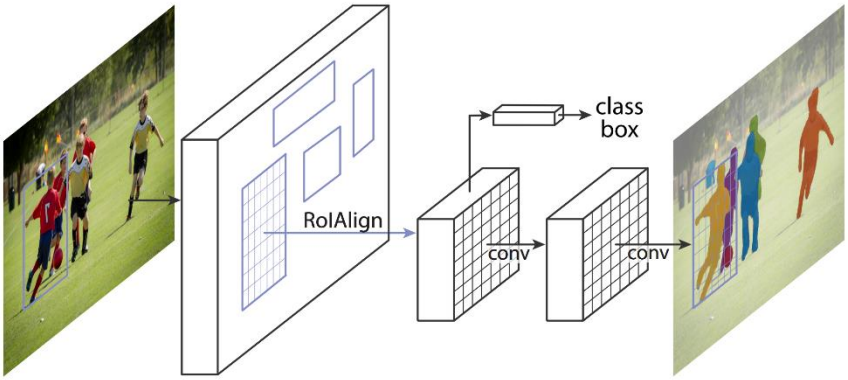

Imagen 14. Estructura de Mask R-CNN. Obtenida en el blog[10].

3.7. Frameworks para aprendizaje profundo

El alto incremento en la demanda de aplicaciones de deep learning y computer vision han provocado que las grandes compañías tecnológicas apuesten por este campo, generando datasets de uso público y buscando crear herramientas que faciliten el desarrollo de estas aplicaciones, ya que la programación a bajo nivel de las arquitecturas de redes neuronales explicadas en la sección anterior va siendo más difícil cuanto más compleja es la solución de estas. Es por ello que se generaliza el empleo de frameworks, entornos de desarrollo diseñados con la idea de facilitar y agilizar el aprendizaje de estas herramientas, además de implantar unos patrones de buenas prácticas que ayudarán a crear un código más ordenado y limpio, consiguiendo así atraer a cada vez más desarrolladores.

En 2002 es lanzado el framework open source conocido como *Torch*, basado en el lenguaje de script conocido como Lua y que cuenta con un extenso soporte para algoritmos de machine learning [35]. Terminó su desarrollo en 2017, cuando apareció su sucesor, *PyTorch*, implementado por Facebook, que introduce el lenguaje de programación Python para ser así más accesible a nuevos desarrolladores [36].

Google compartió en 2015 el framework *Tensorflow*, el cual cuenta con una librería open source para computación numérica que facilita las tareas relacionadas con el aprendizaje supervisado,

[10] https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761e238b46

tales como la obtención de datos y modelos para los entrenamientos, y el refinamiento de los resultados de estos. Admite los lenguajes de programación Python, C++ y R. Goza también de una gran comunidad de desarrolladores, lo que supone la existencia de una gran cantidad de contribuciones en la red [37].

Caffe2 es un framework introducido por Facebook en 2017 como sucesor de Caffe, el cual cayó en desuso debido a la pobre documentación existente y a su difícil compilación [38]. Caffe2 se encuentra enfocado tanto para la implementación en móviles como para entornos de producción a gran escala, haciendo uso de una interfaz en Python con C++ a bajo nivel. Es más escalable y ligero que el otro framework de Facebook, PyTorch [39].

Por otro lado, se encuentra el kit de herramientas de Intel, *OpenVINO*, que se enfoca al campo de la visión por computador y que busca, apoyándose en el hardware específico de Intel, la optimización en la ejecución de redes ya entrenadas. Esta es una herramienta muy versátil ya que es compatible con los frameworks más importantes, como Caffe o Tensorflow, permitiendo la conversión de estos al formato permitido por OpenVINO, conocido como IR. Además, su sitio web ofrece mucha documentación, aplicaciones y una alta gama de modelos preentrenados con distintas arquitecturas de computer vision con formato compatible para los distintos frameworks [40].

En este proyecto se van a tratar una API de Tensorflow para la detección de objetos y el kit de herramientas OpenVINO para el análisis de la velocidad de ejecución de inferencias.

3.8. Hardware para IA

Uno de los puntos más importantes en los procesos de tareas de aprendizaje automático o visión por computador es el hardware empleado, dado que cuando los conjuntos de datos utilizados aumentan en tamaño, estos procesos pueden pasar de tardar minutos a horas, días o, incluso, semanas. Por tanto, los investigadores han de utilizar el hardware más potente, pero siempre teniendo en cuenta el sobrecoste que supondrá, por lo que tendrán que buscar una armonía para adecuarse así a sus posibilidades.

Históricamente, las unidades centrales de procesamiento (CPU), diseñadas por Intel en los años 70 han sido las más empleadas hasta que Nvidia integró la arquitectura CUDA para sus unidades gráficas de procesamiento (GPU) que, si ya bien estaban optimizadas para hacer uso de un alto rendimiento y mucho poder computacional, al introducir esta nueva arquitectura, la cual es una plataforma de computación paralela, se ha conseguido acelerar el rendimiento de las GPUs de manera drástica [41]. Mientras las CPUs hacen empleo de entre 4 y 16 núcleos, con CUDA se llegan a emplear cientos. Como contrapartida, las GPUs, pese a que son más rápidas que las CPUs,

son mucho más caras que estas, lo que puede ser de bastante importancia a la hora de elegir dispositivo.

Las CPUs y las GPUs son empleadas en problemas de computer vision para la ejecución tanto del entrenamiento como de la inferencia. Las desventajas de estas unidades de procesamiento son tanto el coste energético que requieren como su precio.

Con la idea de mejorar la ejecución de estas unidades de procesamiento, surgen librerías matemáticas con este objetivo. Entre ellas se encuentra la librería cuDNN (CUDA Deep Neural Network) de Nvidia, que proporciona a las GPU una aceleración para operaciones con redes neuronales profundas, y obtiene mejores resultados al ser empleada sobre frameworks como los explicados en la sección anterior [42]. Cabe destacar la librería MKL (Math Kernel Library) de Intel, la cual dispone de una alta optimización sobre procesadores de Intel [43].

Más recientemente han surgido distintas herramientas que han permitido que tareas de alto coste computacional lleguen a manos de todos, suponiendo incluso a los desarrolladores evitar el sobrecoste de emplear una GPU. Estas herramientas, diseñadas por las grandes tecnológicas, se pueden diferenciar entre las que hacen empleo de plataformas en la nube, y las que usan pequeños dispositivos hardware, también conocidos como aceleradores de inferencia.

Entre el primer grupo destacan, entre otras, Google Cloud [44], Microsoft Azure Machine Learning [45] o Amazon Web Service [46]. Estas permiten a los usuarios hacer empleo de distintas instancias con potencia dispar (cuanto mayor potencia mayor será el precio a pagar). A parte de compartir herramientas para la ejecución de entrenamientos e inferencias, disponen de servicios de etiquetado de datos, máquinas virtuales epecíficas para tareas de deep learning, etcétera. La principal carencia de estas herramientas surge cuando se quieren emplear para procesar vídeos a tiempo real debido a la baja latencia de red [47].

En el segundo grupo se encuentra la tecnología conocida como Edge AI, que es empleada en la fase de inferencia. Esta busca apoyarse en hardware específico, evitando conexiones online, para obtener así resultados muy superiores en tareas que requieren de procesamiento a tiempo real [47]. El hardware del que hace empleo puede consistir en CPUs y GPUs, aunque para 2025 se ha estimado que el 75% del mercado estará copado por los ASIC (siglas en inglés de circuito integrado para aplicaciones específicas), que son circuitos integrados para una labor en concreta, en este caso, labores de inteligencia artificial [48]. Estos circuitos cuentan con un muy bajo coste energético y disminuido tamaño, lo cual les hace perfectos para ser utilizados para tareas de IoT. Por ejemplo, Google hace empleo del coprocesador Edge TPU [49], que forma el núcleo de los productos de Coral, entre los que se encuentra un USB acelerador que permite ejecutar las inferencias a alta velocidad, y hace uso de los frameworks y bibliotecas Tensorflow, Scikit-learn, XGBoost y Keras [50]. Intel también distribuye este tipo de chips con circuitos integrados. Desde

que adquirió la compañía Movidius en 2016, Intel ha lanzado al mercado las conocidas como Intel Neuronal Computer Stick, que hacen empleo de la unidad de procesamiento de visión (VPU) Intel® Movidius™ Myriad™ X [51]. Al igual que los productos de Coral, esta herramienta está diseñada para ejecutar inferencias, y hace uso del conjunto de herramientas OpenVino™, también de Intel.

4. Metodología

En esta sección se van a explicar los distintos pasos y experimentos que se han llevado a cabo para poner en marcha este proyecto, dividiéndose en dos secciones diferenciadas. La primera se enfoca en un estudio de las técnicas de tratamiento de los conjuntos de imágenes y de evaluación de los resultados. En la segunda se explican los pasos llevados para la configuración y ejecución de la transferencia de aprendizaje.

4.1. Diseño

Para poder poner a punto el entorno de desarrollo, es necesario explicar las distintas técnicas que se van a seguir en el tratamiento de los conjuntos de imágenes en tareas de detección de objetos para su correcta anotación y su posterior aumento de datos, así como las métricas de evaluación que van a ser empleadas.

4.1.1. Tratamiento de los conjuntos de imágenes anotadas

Como se explica en el punto 3.5, los proyectos que utilizan técnicas de computer vision requieren de conjuntos de imágenes anotadas, también conocidos como datasets. Dependiendo de la técnica de visión por computador empleada, el dataset presentará una estructura diferente.

La técnica empleada en este proyecto es la conocida como detección de objetos. Para esta, se hará uso de una estructura formada por pares de imágenes-anotaciones. Esto es, para cada imagen existe un archivo que, para cada cuadro delimitador contenido en esta, cuenta con cuatro variables que indicarán los cuatro puntos que delimitan el cuadro y el nombre de la clase asignada.

Dado que no existe un formato estandarizado, se pueden emplear distintos formatos en función del modelo de detección de objetos empleado, existiendo diversas opciones para esta catalogación. Entre estos destacan COCO, YOLO y Pascal VOC. A continuación, se explican los dos últimos formatos haciendo uso de la imagen 15, en la que aparecen dos objetos, un plátano y una manzana, con los bounding boxes correspondientes superpuestos.

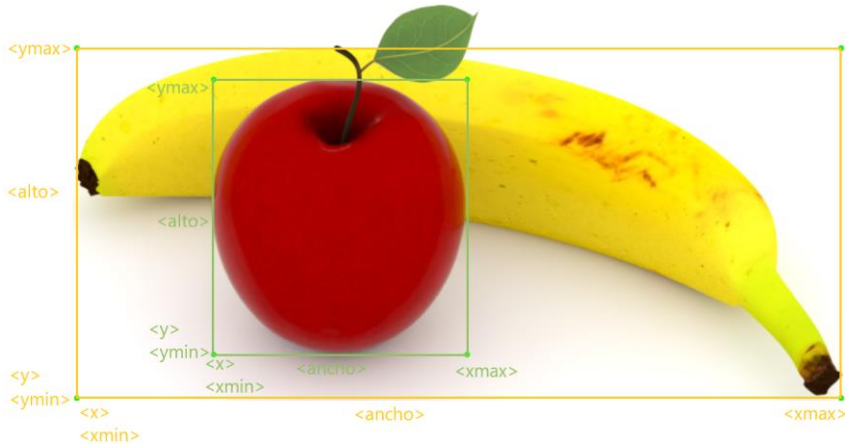

Imagen 15. Ejemplo de bounding boxes para una imagen de un plátano y una manzana. Se pueden ver las etiquetas correspondientes a los formatos YOLO y Pascal VOC. Obtenida en la página[11].

El primer formato, YOLO, hace uso de archivos con extensión .txt para cada imagen, y de un archivo denominado *"classes.txt"* en el que se guardarán las clases empleadas en todo el dataset, siendo su posición en este archivo igual a su identificador numérico.

1 *apple*
2 *banana*

El archivo de texto relacionado con la imagen guarda en cada línea los datos que referencian a un cuadro delimitador. El formato concreto es el siguiente:

1 0.345000 0.455556 0.278889 0.300000

2 0.476667 0.462222 0.837778 0.382222

El primer campo, <id-clase>, se corresponderá al número identificador de la clase a la que hace referencia el bounding box. Los campos <x> e <y> se refieren al punto del cuadro más cercano al eje de coordenadas. Y <ancho> y <alto> se corresponden a la base y la altura del cuadro, que sumándose a <x> e <y> se pueden usar para hallar el resto de vértices que conforman el bounding box.

<id-clase> <x> <y> <ancho> <alto>

Por otro lado, el formato Pascal VOC emplea archivos con el lenguaje de marcado .xml, empleando etiquetas para señalar los parámetros y clase de un cuadro. El formato concreto será el que se ve a continuación, en la imagen 16.

[11] https://es.3dexport.com/3dmodel-banana-and-apple-225048.htm

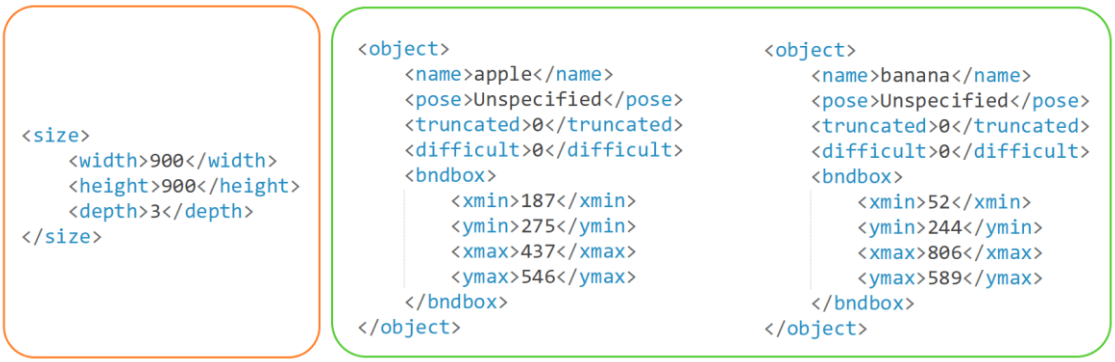

Imagen 16. Formato de Pascal VOC. Se pueden ver la etiqueta <size> (izquierda) que indica los tamaño, en píxeles, de la imagen. También se pueden ver los dos objetos (derecha) bajo las etiquetas <object>.

En la etiqueta <size> (izquierda) se pueden ver las dimensiones (en píxeles) de la imagen. En <object> (derecha) se referencia un único cuadro delimitador, existiendo tantas etiquetas <object> como objetos existan en la imagen. Dentro de esta etiqueta se ven dos etiquetas principales. Una de ellas es <name>, donde estará escrito el nombre de la clase del objeto. La otra es <bndbox> (bounding box), donde están escritos los puntos exactos donde se sitúan los vértices para un cuadro concreto.

4.1.2. Conjuntos estandarizados de imágenes

Haciendo una simple búsqueda en Google se pueden encontrar diversos datasets de código abierto. Las plataformas más importantes en este sector suministran grandes conjuntos de imágenes, los cuales pueden llegar a contar con hasta cientos de miles de imágenes y sirven de estándar para realizar pruebas de clasificación, detección y segmentación. Estos, a su vez, facilitan herramientas para que los usuarios puedan evaluar sus modelos de entrenamiento. Estas grandes plataformas de datos organizan también las conocidas como competiciones o desafíos del computer vision, donde cada una propone a los desarrolladores participantes una serie de retos, enfocados a distintas técnicas de computer vision, y evalúa sus resultados haciendo uso de sus propias métricas de evaluación. Estas plataformas son las siguientes:

- Pascal VOC (Virtual Object Classes), cuenta con un dataset que se ha ido incrementando a lo largo de los años, formado por 11.5 mil imágenes, 27 mil objetos y 6.9 mil segmentaciones, anotados en 20 categorías. Se emplea tanto para clasificación, detección como segmentación. El primer artículo de este desafío fue publicado en 2005, y se hicieron publicaciones anuales con nuevos desafíos y aumentando el tamaño de los datatasets hasta 2012 [52].
- ImageNet, se encuentra organizado de acuerdo a la jerarquía de WorldNet, guardando miles de imágenes según la categoría con la que se relacionan. Es un buen dataset para problemas de clasificación [24].

- Places, es una base de datos desarrollada por el MIT Computer Science and Artificial Intelligence Laboraty, que cuenta con 205 categorías relacionadas mediante etiquetas con 2.5 millones de imágenes. Al igual que ImageNet, es interesante su uso para clasificación [53].
- Open Images de Google, dataset empleado en la clasificación, que hace uso de 9 millones de URLs a imágenes que han sido anotadas a mano mediante etiquetas en un total de más de 6 mil categorías, habiendo imágenes que disponen de hasta 8 etiquetas [54].
- VisualGenome es un dataset que va más alla de la detección de objetos conveciones, bucando crear un contexto en base a la relación que existe entre los distintos objetos que aparecen en una imagen. Para esto hace empleo de 108 mil imágenes, 2.8 millones de objetos, 2.3 millones de relaciones y 5.4 millones de descripciones de regiones [55].
- Microsoft COCO (Common Objects in Context), presentado en 2014, hace empleo de 330 mil imágenes y 1.5 millones de objetos anotados en 80 categorías. Se emplea tanto en problemas de detección de objetos como de segmentación [56].

4.1.3. Herramientas de anotación de las imágenes

A parte de estos datasets compartidos, existen diversas herramientas para poder construir un dataset en busca de poder crear un modelo detección custom. Estas se recomiendan en función de la técnica de computer vision usada, el formato generado como output, el sistema operativo que se dispone y la velocidad de anotación de la misma herramienta.

Entre ellas, la más completa que se encuentra es Lionbridge AI, que junta todas las mejores herramientas de anotación en una, sirviendo así para tareas de clasificación, detección (generando bounding boxes en 2D y 3D), segmentación semántica, etcétera, y cuenta con más 500 mil contribuciones [57]. El único problema que se le encuentra a la herramienta es que es muy pesada, y ofrece mucho más de lo que se requiere para este proyecto.

Por esta razón, se han investigado otras herramientas, decantándose al final por LabelImg para llevar a cabo la tarea. Esta herramienta es específica para la detección de objetos y su uso es muy sencillo. Permite realizar anotaciones en forma de bounding boxes de una forma ágil y eficaz, en formatos YOLO (.txt) y Pascal VOC (.xml). Para usarla basta con clonar el repositorio localizado en [58].

4.1.4. División del conjunto

El siguiente paso en el proceso de aprendizaje de modelos en técnicas de computer vision es realizar una división del conjunto de imágenes entre los datos (imagen junto a su anotación) que serán empleados para la evaluación y los que serán empleados en el entrenamiento.

Al entrenar el modelo se busca que este pueda generalizar correctamente el conocimiento aprendido en el proceso. Un sobreentrenamiento generará un sobreajuste del modelo (del inglés, overfitting), una mala generalización, lo que quiere decir que el modelo no será capaz de encontrar patrones que no se encontrasen previamente en el conjunto de datos empleados para el entrenamiento. Es por esto por lo que del conjunto principal se extrae un pequeño subconjunto, al que se refiere como conjunto de evaluación, que servirá para ir evaluando en paralelo los resultados del entrenamiento llevado a cabo con el conjunto restante.

A la hora de dividir el dataset, se emplean distintos porcentajes para los subconjuntos resultantes. Por norma general, se suele hacer empleo del principio de Pareto, es decir, un 20:80, para la evaluación y el entrenamiento, respectivamente. Esto varía dependiendo del tamaño del dataset utilizado. Cuando el conjunto de imágenes anotadas empleado es muy pequeño, se reduce el porcentaje del conjunto de evaluación, llegando a emplearse una proporción de un 10:90.

También se puede realizar la división en tres partes, añadiendo un subconjunto nuevo conocido como validación. Este se emplea para realizar las evaluciones intermedias llevadas a cabo antes de terminar el entrenamiento. Cuando se hace esta división, el subconjunto de evaluación se emplea tan solo para evaluar el modelo cuando el entrenamiento ha finalizado.

4.1.5. Métricas de evaluación

Para el correcto diseño de un modelo de detección, es importante hacer empleo de distintas métricas de evaluación genéricas para poder así calcular la precisión con la que este detecta los objetos requeridos. A día de hoy, la métrica estándar para detección de objetos es la precisión media promedio (del inglés, mean average precision), y se suele referir a ella como mAP. Otra medida que se emplea es la conocida como recuperación media promedio, mAR [59]. Para entender mejor qué son estas medidas, es interesante entender primero los siguientes conceptos.

El primer concepto a tener en cuenta es la medida conocida como *"Intersection over Union"* (IoU), la cual, como su nombre indica, resulta de dividir el área de la intersección del cuadro delimitado real (al que se conoce también como ground truth), creado a mano y que hace una referencia real al objeto, y el del cuadro generado en la predicción, entre el área total de ambos cuadros.

Imagen 17. Ejemplificación de distintos IoU en función de lo estrictas que son las intersecciones entre los ground truth (verde) y los bounding box predichos (rojos). Obtenida en la página[12].

El resultado de esta división será un numero k, que se encontrará entre 0 y 1. Cuanto más cercano a 1 sea, más precisa será la predicción. Dado que es muy difícil, por no decir imposible, conseguir un IoU igual a 1, es decir, que ambos cuadros coincidan completamente, se tiende a establecer un valor mínimo, que suele coincidir con 0.50, que nos informará como de buena es la detección. Dependiendo del problema a resolver, será necesario establecer un menor o mayor grado de coincidencia entre los recuadros.

$$IoU = \frac{\text{área del ground truth} \cap \text{área de la predicción}}{\text{área del ground truth} \cup \text{área de la predicción}}$$

La necesidad de emplear un IoU menos estricto se hace más notable cuando los objetos a detectar son de tamaño reducido. Por ejemplo, para dos imágenes de mandarinas en las que los ground truth de estas tienen distintos tamaños, 7x7 y 50x50 píxeles, los bounding box detectados tienen el mismo tamaño que los anteriores, pero se encuentran desplazados 3 píxeles a la derecha (eje X) y otros 2 hacia arriba (eje Y). Al realizar la ecuación para obtener el IoU resultará que, pese a haber errado en la misma cantidad de píxeles, el IoU para el objeto grande será muy superior.

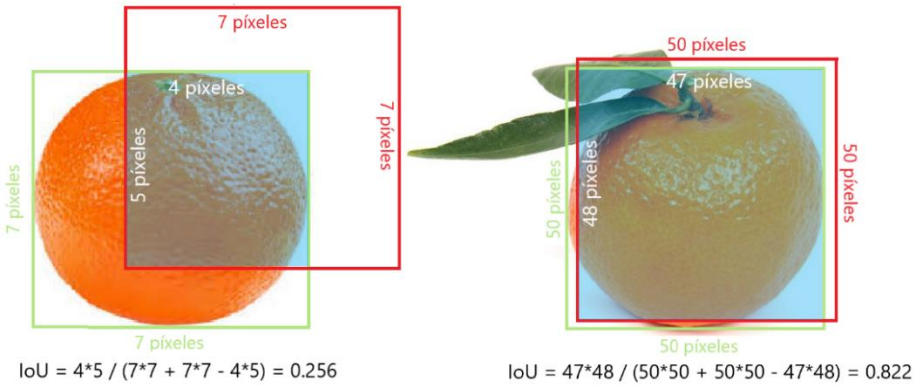

Imagen 18. A la izquierda se puede ver una mandarina de 7x7 píxeles. A la derecha otra mandarina de 50x50 píxeles. Se observa la diferencia de IoU entre ambos objetos al fallar la predicción en una misma cantidad de píxeles. Imágenes obtenidas en los siguientes enlaces [13] y [14].

[12] https://www.freepnglogos.com/pics/car
[13] https://www.frutality.es/project/beneficios-de-la-mandarina-%F0%9F%8D%8A/
[14] https://actualfruveg.com/2018/10/31/mandarina/

Una vez conocida la definición de IoU, se puede explicar el significado de las siguientes cuatro medidas, que hacen referencia a si una detección es o no correcta.

- True Positive (TP). Una detección será TP si se cumple las condiciones de que la clase predicha coincide con la clase a la que pertenece el ground truth, que la puntuación en la detección es mayor que un umbral, y que el IoU es superior al establecido, como por ejemplo, 0.50 o 0.75.
- False Positive (FP). Una detección será FP si la primera o la última condición mencionadas en el apartado de TP se incumplen.
- False Negative (FN). Esto se dará cuando la puntuación de la detección sea inferior al umbral, pese a que en este caso el IoU sea superior al umbral elegido.

Una vez explicadas las anteriores medidas, se puede explicar qué es la medida *"precisión"*. Esta es el resultado al dividir los TP entre el conjunto de las detecciones consideradas positivas, o sea, la suma de TP y FP.

$$Precisión = \frac{TP}{TP + FP}$$

Por último, hay que mencionar la medida conocida como *"recall"* (recuperación), que surge de la división de los TP entre el conjunto de detecciones acertadas, es decir, las suma de TP y FN.

$$Recuperación = \frac{TP}{TP + FN}$$

La precisión media (AP) es igual al área que se encuentra bajo la curva de pares de precisión (eje Y) y recall (eje X) surgidos al cambiar el umbral de detección. La AP obtendrá un resultado diferente para cada una de las clases que se encuentran en la imagen, usándonse la métrica mAP para hallar el promedio de AP para todas las clases.

$$mAP = \frac{\left(\sum_{i=1}^{k} AP\ i\right)}{K}$$

De igual manera, la mAR es igual al promedio de AR para todas las clases. La AR (average recall) es el promedio para cada clase del *"recall"* para todas las medidas IoU entre 0.50 y 0.90.

$$mAR = \frac{\left(\sum_{i=1}^{k} AR\ i\right)}{K}$$

4.1.6. Métricas de los desafíos de evaluación

Las métricas explicadas en el apartado anterior han sido estandarizadas y empleadas en los conocidos como desafíos o competiciones para la evaluación en la detección de objetos, ya

mencionados en el capítulo 4.1.2, como Pascal VOC, COCO o ImageNet. La precisión para cada competición varía en función del umbral de IoU empleado. Por ejemplo, las métricas de PASCAL VOC usan un IoU = 0.50, por tanto, los bounding boxes detectados cuya coincidencia con sus respectivos ground truth sea menor a un 50% no serán tenidos en cuenta de cara a la evaluación. Como ya se explicado, se considera que la métrica de evaluación es más o menos estricta en función del IoU empleado, siendo más estricto cuanto más se acercano se encuentre a 1.00.

El desafío de COCO hace uso de 6 métricas para calcular el mAP, que varían entre las que catalogan por el tamaño de los objetos detectados, y las que se fijan en el umbral de deteccion empleado [60]. A continuación, se puede ver una gráfica proporcionada por COCO en la que se exponen las distintas métricas que emplea.

```
Average Precision (AP):
   AP                  % AP para IoU=.50:.05:.95 (métrica primaria del desafio)
   AP^IoU=.50          % AP para IoU=.50 (métrica de PASCAL VOC)
   AP^IoU=.75          % AP para IoU=.75 (métrica estricta)
AP por tamaño de los objetos:
   AP^small            % AP para objetos pequeños: area < 32^2
   AP^medium           % AP para objetos medianos: 32^2 < area < 96^2
   AP^large            % AP para objetos grandes: area > 96^2
Average Recall (AR):
   AR^max=1            % AR dada 1 detección por imagen
   AR^max=10           % AR dadas 10 detecciones por imagen
   AR^max=100          % AR dadas 100 detecciones por imagen
AR por tamaño de los objetos:
   AR^small            % AR para objetos pequeños: area < 32^2
   AR^medium           % AR para objetos medianos: 32^2 < area < 96^2
   AR^large            % AR para objetos grandes: area > 96^2
```

Imagen 19. Medidas empleadas en la evaluación de COCO. Obtenida en la página[15].

En esta gráfica AP se corresponde a mAP. Por ejemplo, en la primera métrica, que es la métrica que define COCO, "*AP para IoU=.50:.05:.95*", se hace una media de las AP de todas las clases detectadas con distintos IoU, con valores múltiplos de 5 desde 0.50 a 0.95, es decir, con IoU = {0.50, 0.55, 0.60, …, 0.95}. Los resultados de COCO son interesantes, porque a parte de ofrecer los resultados de esta métrica, se obtienen los de Pascal VOC.

4.2. Implementación de la transferencia de aprendizaje

En esta sección se explican minuciosamente los pasos seguidos para implementar el proceso de detección de mandarinas haciendo uso de un dataset propio y las métricas de evaluación que cumplen con los puntos explicados en el apartado de diseño. Estos se pueden dividir en dos partes, consistiendo la primera en obtener y preparar los conjuntos de imágenes (haciendo uso de una herramienta de anotación, sumada a la división del conjunto y el empleo de técnicas de aumento de datos), y configurar la herramienta que se va a emplear para la detección. La segunda, en cambio, se enfocará en explicar la transferencia de aprendizaje para realizar los entrenamientos

[15] http://cocodataset.org/#detection-eval

de los distintos experimentos. Se hará empleo de una API para detección de objetos de Tensorflow, que emplea programas en lenguaje Python, el cual es un lenguaje interpretado y multiplataforma, que se encuentra disponible para los sistemas operativos Windows, Mac y Linux. En el esquema que se encuentra en la imagen siguiente se puede ver una representación de este proceso.

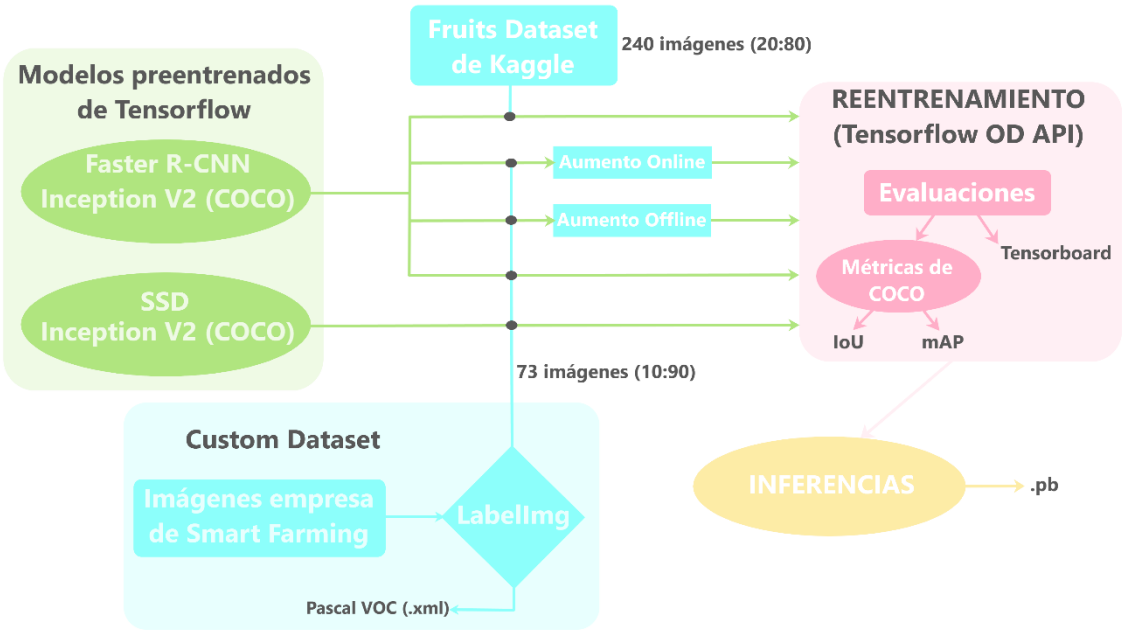

Imagen 20. Esquema de la metodología seguida para llevar a cabo la transferencia de aprendizaje.

En este proceso se han de tener en cuenta primero las especificaciones de hardware del equipo de trabajo, las cuales se pueden ver a continuación:

- Sistema Operativo: Windows 10 Pro 64 bits
- Procesador CPU: Intel® Core™ i7-7660U CPU 250GHz (4 CPUs)
- Memoria RAM: 8 GB

Para trabajos que hacen empleo de modelos de aprendizaje profundo, estas especificaciones son de vital importancia ya que los procesos llevados a cabo hacen un uso exhaustivo de los recursos del equipo llegando a ponerlo al límite y resultando estos, en ocasiones, insuficientes para llevar a cabo el proceso. Esto puede llegar a suponer tener que cambiar de rumbo en el proyecto o tener que realizar un desglose de presupuesto para conseguir una herramienta con mejores especificaciones técnicas.

Por otro lado, es también interesante comentar las distintas herramientas y librerías empleadas en esta labor. Entre ellas se encuentran las siguientes (entre paréntesis la versión exacta empleada):

- Python (3.7.4)
- TensorFlow (1.15.0)

- Jupyter (Anaconda)
- pip (19.3)
- Pycocotools (2.0)
- Numpy (1.17.2)
- Cython (0.29.13)
- Matplotlib (3.1.1)
- Pandas (0.25.1)
- Pillow (6.2.0)
- Opencv-python (4.1.1.26)
- Scikit-image (0.16.2)
- LabelImg

4.2.1. Obtención de imágenes y técnica de anotación de estas

Como se ha comentado previamente, el proceso de entrenamiento de modelos de Deep Learning para técnicas de computer vision hace uso de conjuntos imágenes anotadas, también conocidos como datasets. En este trabajo, en el que se busca la detección de objetos en imágenes, para cada imagen del conjunto existirá un archivo .xml en formato Pascal VOC (Visual Object Classes).

Para llevar a cabo esta tarea se necesita primero un conjunto de imágenes que cumpla la premisa buscada: poder detectar mandarinas maduras (de color naranja) y no maduras (de color verde) colgadas de un árbol.

Se cuenta desde el comienzo con un conjunto de imágenes no anotadas proporcionadas por la empresa IT enfocada interesada en el proyecto, consistente en 73 imágenes de mandarinos con unas dimensiones de 3120x4160 píxeles. En esta se aprecian mandarinas en distintos estados de maduración, siendo en ocasiones muy complicado detectar visualmente las mandarinas que se encuentran en fases tempranas de maduración (verdes), al mimetizarse con el árbol en el que se encuentran.

Pero antes de trabajar directamente con ese conjunto de imágenes, y como toma de contacto, se realiza una búsqueda en la web de datasets que puedan valer para este trabajo. Se da con un dataset de frutas, facilitado por Kaggle, plataforma online de investigación, propiedad de Google [61].

Este dataset consiste en 300 imágenes anotadas en formato Pascal VOC, y las clases anotadas son “apple”, “banana”, “mixed” y “orange”. El tamaño de las imágenes varía entre los 300x300 y los 1200x1200 píxeles, ocupando los objetos que se encuentran en su interior casi la totalidad de la imagen. Dado que lo que se está buscando es que se detecten mandarinas colgadas del árbol que les da nombre, y debido a que las naranjas tienen un aspecto bastante similar a las mandarinas, se

emplea este primer dataset para generar un inferencia que después se pueda emplear en la evaluación de distintas imágenes pertenecientes al conjunto suministrado por la empresa de Smart Farming con el fin de probar si es capaz de detectar las mandarinas como naranjas.

Por otro lado, en busca de obtener una mayor precisión se decide generar un dataset propio en base a las 73 imágenes suministradas por la empresa IT. Para esta labor se hace empleo de la herramienta de anotación open source conocida como LabelImg. Como se ha mencionado antes, esta es una de las más empleadas en proyectos de detección de objetos, siendo altamente recomendada para este tipo de tarea ya no solo por la facilidad de su instalación y uso, sino también porque cuenta con bastante soporte gracias a su gran base usuarios.

Al emplear en el entrenamiento y la evaluación este conjunto de imágenes anotadas, los resultados pueden ser bastante mejores al compartir el contexto en cuanto a ruido, calidad y tamaño de los objetos.

Antes de comenzar con el proceso de anotación se decide reducir el tamaño de estas imágenes que, como se ha indicado antes, tenían una resolución de 3120x4160, siendo el tamaño de las mandarinas que se encuentran en su interior de un tamaño de más o menos 180x180 píxeles. El tamaño resultante de las imágenes será igual a 600x800, y el de las mandarinas de unos 35x35 píxeles. Esta reducción de tamaño conseguirá evitar errores en el entrenamiento debido a problemas derivados de la falta de memoria durante el entrenamiento, aunque el reducido tamaño de las mandarinas supondrá una disminución de la precisión en la detección para las que cuenten con mejor resolución.

Durante la anotación de las mandarinas se dividen estas en dos grupos atendiendo a su estado de maduración, clasificándolas según su color como “mandarinas naranjas” o “mandarinas verdes”. Este dataset se encuentra localizado en [62].

Esta tarea se hace muy tediosa ya que, pese a que el conjunto de imágenes no es muy extenso, existe una considerable cantidad de mandarinas (objetos) por imagen, que suman un total 2206 entre todas las imágenes.

4.2.2. Divión del dataset

Para los dos conjuntos empleados, al no coincidir en tamaño, se emplearán distintos porcentajes evaluación/entrenamiento. La división se realiza a mano, procurando que las imágenes que componen los conjuntos de evaluación supongan una muestra representativa de los conjuntos principales. La API de Tensorflow Object Detection no requiere de un subconjunto de validación por lo que se contará tan solo con dos subconjuntos, el de entrenamiento y el de evaluación para evaluar los resultados periódicamente:

- Para el dataset de Kaggle se empleará el principio mencionado anteriormente, eligiendo para la evaluación un 20% de los datos (60 imágenes para ser concretos), y un 80% para el entrenamiento (240 imágenes).
- Para el dataset custom, al ser de menor tamaño (73 imágenes), se ha decidido disminuir el tamaño del conjunto de evaluación al 10% (15 imágenes) para así tener una mayor cantidad de datos en el conjunto de entrenamiento, siendo este de un 90% del tamaño del dataset original (58 imágenes).

4.2.3. Aumento de datos

El empleo de esta técnica es uno de lo más comunes para mejorar el proceso de entrenamiento, y sirve también para reducir el problema conocido como sobreajuste. Consiste en crear nuevas imágenes anotadas al realizar modificaciones puntuales en los datos del conjunto original. Estas modificaciones pueden consistir en rotar la imagen por ángulos (90º, 180º y 270º), voltear la imagen horizontal y verticalmente, escalado, recorte, aumento del color de pca, modificar la saturación, etcétera [63].

Por otro lado, existen dos formas de llevar a cabo esta técnica:

1- De forma manual o local (en inglés, offline). Esto es, aumentando el dataset local, haciendo uso de diferentes scripts para aumentar la cantidad de imágenes y de sus anotaciones. Existen dos modos de llevar realizar este aumento: se puede ampliar el conjunto de imágenes sin anotar, para después anotar todas las imágenes, tanto las viejas como las nuevas, una a una; o ampliar el conjunto de las imágenes ya anotadas mediante el uso de scripts que permitan modificar también las anotaciones para que se correspondan a las nuevas imágenes resultantes tras la ampliación.

   La mayor desventaja del aumento de datos offline es que esta técnica hace uso de una gran cantidad de espacio en el disco, lo cual puede llegar a ser bastante problemático cuando no se cuenta cuenta con espacio suficiente, por no mencionar la amplia cantidad de tiempo que lleva la elaboración de este aumento.

2- Haciendo uso de un framework que permita el uso de la técnica conocida como “online data augmentation”, las cuales realizan aumentos sobre la marcha en el mismo entrenamiento, buscando así conseguir una disminución en cuanto a la memoria empleada, aunque esto puede derivar en una reducción en la velocidad del entrenamiento.

En este proyecto se ha hecho empleo de ambas técnicas de aumento. En cuanto al aumento de datos online, los distintos modelos que han sido empleados en el entrenamiento, como se explica a continuación, vienen preconfigurados para emplear algunas técnicas de data augmentation

mediante llamadas a distintos métodos, y es posible añadir técnicas extra al entrenamiento al modificar el archivo de configuración del modelo empleado para el entrenamiento.

Por otro lado, se ha hecho uso de los métodos de aumento offline compartidos en el blog [64], modificando varios parámetros para adecuarlos a nuestro sistema operativo, para aumentar así el conjunto de datos localmente. Estos métdos consisten tres archivos en lenguaje de programación Python que, tras ser debidamente ejecutados sobre un conjunto de imágenes ya anotadas, generan un aumento de factor x4, es decir, por cada imagen anotada se obtienen otras 3 nuevas. Estos nuevos datos resultarán de realizar las siguientes transformaciones al conjunto original: rotación de 90º, volteado horizontal y vertical.

Imagen 21. Distintos ejemplos de aumento de datos. Se pueden ver, de izquierda a derecha, la imagen real de un mandarino, la misma imagen rotada 90º, volteada horizontalmente y volteada verticalmente.

4.2.4. Preparación de Tensorflow Object Detection API

Tras haber preparado correctamente el dataset, este se puede entrenar haciendo uso de distintos modelos ya preentrenados con los datasets de COCO, Kitti, Open Images, Ava v2.1, Pixel4 Edge, Mobile e iNaturalist Species Detection. Como ya se mencionó antes, la transeferencia de aprendizaje es una técnica del machine learning que ayuda a reducir los tiempos de entrenamiento para datasets pequeños, aumentando la precisión y reduciendo el coste computacional del proceso, ya que ha sido preentrenado con enormes conjuntos de imágenes generalizadas para este propósito. Para llevar esta transferencia a cabo se hace empleo de la técnica conocida como fine-tuning (ajuste fino), que consiste en realizar un ajuste de la red a partir de los pesos ya inicializados del modelo preentrenado congelado, también conocido como inferencia, para que se puedan emplear en la nueva tarea.

Estos modelos preentrenados son facilitados por la herramienta open source conocida como TensorFlow Object Detection API [65], la cual hace empleo de la biblioteca Tensorflow para ayudar a los usuarios a construir e implementar modelos de detección de objetos de un modo bastante sencillo, siendo así una herramienta ideal tanto para los usuarios con experiencia como para los que no la tienen.

El primer paso será clonar el repositorio de esta herramienta, a la altura de *"tensorflow/models"* [66]. A su vez, desde la ruta *"...models/.../g3doc_detecion_model_zoo.md"* se podrán observar, junto al enlace a la descarga de cada modelo (en formato comprimido .tar.gz), la velocidad de ejecución en milisegundos para una imagen de tamaño 600x600 píxeles y la precisión media haciendo uso de la evaluación ofrecida por COCO.

Con estos datos sobre la mesa, se decide descargar dos modelos cuya velocidad de ejecución (también conocida como velocidad de inferencia) sea lo más alta posible, pero manteniendo una tasa de aciertos decente. Elegimos hacer empleo de dos modelos para probar así dos arquitecturas explicadas previamente: SSD y Faster R-CNN, haciendo uso de una red CNN Inception V2 como extractor de características y preentrenados con el dataset COCO. Los modelos concretos son los siguientes:

- *ssd_inception_v2_coco_2018_01_28*
- *faster_rcnn_inception_v2_coco_2018_01_28*

Tras descargar y descomprimir cualquiera de estos modelos, se encuentran los siguientes ficheros en la carpeta resultante:

1- *frozen_inference_graph.pb*, es el modelo preentrenado congelado, preparado para ser empleado en la detección de objetos (coincidentes con las clases con las que ha sido preentrenado) en imágenes. Al reentrenar el modelo, se buscará generar una nueva inferencia, o congelación del modelo, personalizada a nuestro conjunto de imágenes.

2- Un punto de control formado por los archivos: *model.ckpt.data-00000-of-00001*, *model.ckpt.index* y *model.ckpt.meta*. Esto servirá para que, en caso de haber terminado el entrenamiento, podamos volver a entrenar desde el punto en el que se quedó sin tener que volver a entrenar desde cero todo.

3- *pipeline.config*, que es el archivo de configuración del modelo empleado para generar el grafo. Este archivo se puede encontrar también en la carpeta *"object_detection/samples/",* donde se encontrarán todos los archivos de configuración para todos los modelos. Todos estos archivos de configuración tendrán una estructura común, que será siempre similar a la que se ve en la siguiente imagen.

```
model {
  ssd {
    num_classes: 90
    image_resizer { ...}
    feature_extractor { type: "ssd_inception_v2" }
    anchor_generator {
      ssd_anchor_generator {
      ...
      }
    }
    post_processing {
      batch_non_max_suppression {
        score_threshold: 0.300000011921
        iou_threshold: 0.600000023842
        max_detections_per_class: 100
        max_total_detections: 100
      }
      score_converter: SIGMOID
    }
  }
}
train_config {
  batch_size: 24
  data_augmentation_options { }
  optimizer { initial_learning_rate: 0.00400000018999 }
  fine_tune_checkpoint: "PATH_TO_BE_CONFIGURED/model.ckpt"
  num_steps: 200000
}
train_input_reader {
  label_map_path: "PATH_TO_BE_CONFIGURED/mscoco_label_map.pbtxt"
  tf_record_input_reader {
    input_path: "PATH_TO_BE_CONFIGURED/mscoco_train.record"
  }
}
eval_config {
  num_examples: 8000
  max_evals: 10
  use_moving_averages: false
}
eval_input_reader {
  label_map_path: "PATH_TO_BE_CONFIGURED/mscoco_label_map.pbtxt"
  tf_record_input_reader {
    input_path: "PATH_TO_BE_CONFIGURED/mscoco_val.record"
  }
}
```

Imagen 22. Ejemplo generalista del archivo de configuración de un modelo preentrenado facilitado por Tensorflow Object Detection API.

En la ruta *"models/research/object_detection/"* se pegará el dataset que se quiera usar para entrenar. El dataset se encontrará en una carpeta llamada *"images/"*, que contendrá dos subcarpetas, *"test/"* y *"train/"*, que contendrán a su vez los pares de archivos .jpg y .xml de evaluación y entrenamiento, respectivamente. A continuación, se ejecutará el archivo *"xml_to_csv.py"* [67], que unificará todos los archivos .xml en formato Pascal VOC que hay en las carpetas *"test/"* y *"train/"* en dos ficheros en formato .csv, llamados *"test_labels.csv"* y *"train_labels.csv"*. Así, por ejemplo, en *"test_labels.csv"* se tendrá para cada línea el nombre de una imagen de la carpeta *"test/"* junto con la anotación de uno de sus bounding boxes.

```
filename,width,height,class,xmin,ymin,xmax,ymax
apple_77.jpg,300,229,apple,134,23,243,115
apple_77.jpg,300,229,apple,107,126,216,229
apple_77.jpg,300,229,apple,207,138,298,229
apple_78.jpg,350,350,apple,10,8,344,336
apple_79.jpg,0,0,apple,77,183,641,716
apple_80.jpg,600,500,apple,155,105,453,436
apple_81.jpg,1500,1749,apple,70,398,1388,1731
apple_82.jpg,416,470,apple,65,122,365,404
apple_83.jpg,800,745,apple,1,1,794,738
apple_84.jpg,850,565,apple,64,84,484,457
apple_84.jpg,850,565,apple,550,44,850,356
apple_84.jpg,850,565,apple,197,1,559,273
apple_85.jpg,500,400,apple,88,41,415,393
apple_86.jpg,425,282,apple,112,144,173,211
apple_86.jpg,425,282,apple,250,140,322,215
```

Imagen 23. Ejemplo del formato que se obtendrá al convertir Pascal VOC a formato .csv . En la primera línea se ven los nombres de los distintos campos. En las siguientes líneas se muestran los distintos bounding boxes contenidos en las distintas imágenes.

El paso siguiente será convertir estos archivos en formato .csv en otros con formato TFRecord (TensorFlow Record), que almacena una secuencia de registros binarios y está optimizado para el uso con TensorFlow [68]. Para esto se ha de seguir dos pasos.

El primero consistirá en añadir las etiquetas al archivo *"generate_tfrecord.py"*, para después ejecutarlo. Así se crearán en la carpeta *".../object_detection/"* dos archivos en este formato: *"test.record"* y *"train.record"*. Para el dataset de Kaggle, el contenido modificado del archivo será el siguiente:

```
# TO-DO replace this with label map
def class_text_to_int(row_label):
    if row_label == 'apple':
        return 1
    if row_label == 'banana':
        return 2
    if row_label == 'orange':
        return 3
    else:
        return None
```

Imagen 24. Modificación que hay que realizar en "generate_tfrecord.py".

La segunda tarea consiste en crear otra carpeta dentro de *"object_detection/"* a la que llamaremos *"training/"*, en esta se tendrán que insertar dos archivos. El primero será un archivo al que llamaremos *"labelmap.pbtxt"*, en el que escribiremos las clases las clases que aparecen en el dataset. Siguiendo con el dataset de Kaggle, tendremos el siguiente contenido:

```
item {
    id: 1
    name: 'apple'
}
item {
    id: 2
    name: 'banana'
}
item {
    id: 3
    name: 'orange'
}
```

Imagen 25. Contenido de labelmap.pbtxt para el ejemplo de Kaggle.

El otro archivo será el de configuración del modelo deseado (pipeline.config). Como se ha comentado antes, en la carpeta "object_detection/samples/" se encuentran los archivos de configuración de todos los modelos preentrenados de TensorFlow. Aquí se modifican algunas líneas para adaptar las rutas de entrenamiento y evaluación del modelo a nuestro sistema.

```
train_input_reader: {
   tf_record_input_reader { input_path: "C:/…/object_detection/train.record" }
   label_map_path: "C:/…/object_detection/training/labelmap.pbtxt"
}
eval_input_reader: {
   tf_record_input_reader { input_path: "C:/…/object_detection/test.record" }
   label_map_path: "C:/…/object_detection/training/labelmap.pbtxt"
```

A parte de estos cambios genéricos, se realizan modificaciones puntuales en este archivo de configuración, dependiendo del modelo y el dataset empleados, como se explicará en la sección de evaluación. Es interesante conocer que en la ruta *"object_detection/protos/"* se encuentran los

archivos *"preprocessor.proto"* y *"preprocessorpb2.proto"* donde se pueden ver las distintas operaciones de incremento de datos online que se pueden utilizar en los entrenamientos mediante el uso de esta API.

Cuando el archivo de configuración se encuentre completamente adaptado a nuestro conjunto de datos, se llevará a cabo la ejecución del archivo *"model_main.py"* en busca de entrenar el modelo preentrenado escogido. Este es el código utilizado en la segunda versión de la API, siendo el anterior código de ejecución del entrenamiento *"train.py"* (el cual se encuentra en la misma ruta). En este primero se mostraban por terminal la pérdida para todos los pasos dados en el entrenamiento, al contrario que en *"model_main.py"*, donde se muestra la pérdida para el primer paso (1), y después se va mostrando cada 100 pasos (101, 201, ..., 20301). Otra diferencia importante es que en la versión que se ha utilizado se realizan evaluaciones de forma paralela al entrenamiento sin necesidad de ejecutar en otra terminal un archivo python (eval.py), como se hacía con *"train.py"*. El comando para poner en funcionamiento el entrenamiento es el siguiente.

> *C:\...\object_detection>python model_main.py –logtostderr -model_dir=training/ --pipeline_config_path=training/faster_rcnn_inception_v2_coco.config*

Cabe destacar que en la ruta *".../object_detection/"* nos encontraremos un fichero en formato Jupyter Notebook llamado *"object_detection_tutorial.ipynb"*. Este sirve de ejemplo para realizar una detección de las imágenes que se encuentren guardadas en *".../object_detection/test_images/"*. Para emplear nuestras propias inferencias tendremos que sustituir en el apartado "Load a (frozen) Tensorflow model into memory" las primeras líneas por las que se ven a continuación.

```
PATH_TO_FROZEN_GRAPH = 'inference_graph/nuestra_inferencia.pb'
PATH_TO_LABELS = os.path.join('training/labelmap.pbtxt')
```

También será necesario hacer dos modificaciones en la llamada a la función que se realiza para dibujar los bounding boxes predichos *"vis_util.visualize_boxes_and_labels_on_image_array"*. La función de la primera línea será quitar el límite de bounding boxes dibujados, dado que el máximo está prefijado en 20, y las imágenes del conjunto facilitado por la empresa IT contienen de 7 a 119 objetos. En la segunda línea se igualará el tamaño de los bordes a 2, buscando así que la imagen se vea con una mejor calidad.

```
max_boxes_to_draw=None,
line_thickness=2)
```

Para concluir, el esquema final de nuestro proyecto, donde se encuentran las carpetas y ficheros mencionados antes, es el siguiente.

```
object_detection {
    faster_rcnn_inception_v2_coco_2018_01_28 { ... }
    generatetfrecords.py
    images {
        test { ... }
        train { ... }
        test.cv
        train.csv
    }
    inference_graph { frozen_inference_graph.pb }
    model_main.py
    object_detection_tutorial.ipynb
    protos { preprocessor.proto }
    ssd_inception_v2_coco_2018_01_28 { ... }
    samples {
        configs {
            faster_rcnn_inception_v2_coco.config
            ssd_inception_v2_coco.config
        }
    }
    test_images { ... }
    training {
        pipeline.config
        labelmap.pbtxt
    }
    xml_to_csv.py
}
```

Imagen 26. Estructura de nuestro proyecto en el sistema de archivos.

4.2.5. Transferencia de aprendizaje

En esta sección se explica la motivación de los entrenamientos para los distintos experimentos que se han llevado a cabo, centrándonos en las modificaciones específicas realizadas para cada caso, tanto en los archivos de configuración de los modelos preentrenados como en los datasets empleados, y en los tiempos que han necesitado para completarse.

Antes de comenzar, es importante explicar la herramienta que se va a emplear para la correcta evaluación a tiempo real del entrenamiento. Esta es Tensorboard de Tensorflow, que es perfecta para visualizar las funciones que generan los valores de las distintas métricas del entrenamiento que estamos llevando a cabo a lo largo del tiempo. Entre ellas se encuentran las métricas de evaluación que nos facilita la api de COCO, el learning rate y, las más importantes, las funciones de pérdida de entrenamiento y de evaluación, que ayudarán a dictaminar si nuestro modelo está sufriendo overfitting o no. También permite visualizar las imágenes empleadas en la evaluación del entrenamiento, superponiendo los ground truth y los bounding boxes detectados para poder hacer una comprobación visual de los resultados a tiempo real [69].

Esta herramienta se ejecuta paralelamente al entrenamiento abriendo otra terminal y ejecutando el siguiente comando:

tensorboard –logdir=training

- ***Dataset "Fruit" de Kaggle***

Para comenzar, y a modo de prueba, se hace empleo del dataset de frutas facilitado por Kaggle. La idea de esta prueba es que, una vez generada la inferencia, se ejecute esta con el archivo Jupyter Notebook mencionado antes para comprobar visualemente la veracidad de los resultados. Para llevar a cabo esta tarea se utiliza uno de los modelos preentrenados ya mencionado antes, *faster_rcnn_inception_v2_coco_2018_01_28.* Se escoge este debido a que tiene mejor tasa de aciertos que el de arquitectura SSD, y la diferencia de tamaño en píxeles entre las naranjas del dataset Kaggle y las mandarinas que aparecen en el conjunto de imágenes suministradas por la empresa de smart farming sugieren que la tasa de aciertos no va a ser óptima. Es conveniente comentar, de cara a futuras pruebas, que este modelo tiene predefinida una técnica de aumento de datos online, en concreto *"random_horizontal_flip"*, que provocará volteados horizontales en las imágenes de forma aleatoria en un 50% de las ocasiones.

Para realizar el entrenamiento se tienen que modificar distintos parámetros en el archivo de configuración: *"num_classes"* se igualará a 3, las clases que están presentes en el dataset; *"num_examples"* será igual a 60, que es el total de imágenes de evaluación que disponemos; en *"fine_tune_checkpoint"* se pega la ruta en la que se encuentra el checkpoint del modelo preentrenado congelado que hemos descargado.

```
faster_rcnn { num_classes: 3 }
train_config: {
    fine_tune_checkpoint: "C:/.../object_detection/faster_rcnn_inception_v2_coco_2018_01_28/model.ckpt"
}
eval_config: {
    num_examples: 60
}
```

Tras llevar a cabo estas modificaciones se inicializa el entrenamiento haciendo uso de la clase *"model_main.py"*. Tras 7 horas y 4900 pasos entrenando, la función de pérdida de entrenamiento converge a 0.15 y, haciéndo uso de Tensorboard, comprobamos que la función va a mantenerse estable por tanto se decide parar el entrenamiento.

Las medidas de evaluación ofrecidas por las métricas de COCO dan resultados bastante buenos, comprobándose en la sección de imágenes de Tensorboard que las predicciones se corresponden la realidad.

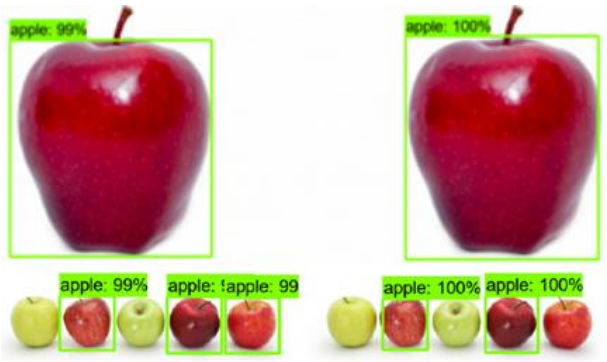

Imagen 27. A la izquierda se ven los bounding boxes predichos y a la derecha los ground truth.

Cuando la tasa de aciertos es la adecuada, se decide terminar el entrenamiento y se genera la inferencia en la carpeta *"inference_graph/kaggle_dataset/faster_coco/"* haciendo uso de la clase *"export_inference_graph.py"*. Ahora se emplea esta inferencia con el archivo *.ipynb* de Jupyter Notebook que facilita con la api. Este archivo se empleará para evaluar 10 imágenes aletorias del conjunto facilitado por la empresa IT, que se situarán en la carpeta *".../object_detection/other_images/"*.

Al ejecutar el cuaderno de Jupyter con nuestro modelo congelado, se comprueba que los resultados no son los deseados. Para una imagen en la que aparecen naranjas, las cuales se ven muy claramente al ser de un color naranja muy pronunciado, se visualizan detecciones, pero de las más de 30 naranjas que aparecen en la imagen se detectan un total de 9, lo cual no promete demasiado. Por otro lado, usando imágenes del conjunto compartido por la empresa, con mandarinas que se distinguen bastante mal, la tasa de detección es muy baja. Estos resultados se pueden ver en la imagen siguiente.

Imagen 28. Resultado de probar el dataset de Kaggle entrenado con Faster R-CNN e Inception V2 con una imagen de un naranjo obtenido en internet (las naranjas se ven muy bien), y dos imágenes pertenecientes al conjunto de imágenes real.

Con la vista puesta en obtener mejores resultados en la detección de las mandarinas, en el punto siguiente se utilizará el dataset personalizado mencionado anteriormente.

- ***Dataset personalizado***

En los siguientes experimentos, con la vista puesta en conseguir una mejor tasa de detección, se va a emplear en el entrenamiento el dataset customizado creado con LabelImg. Como se explicó anteriormente, se va a hacer uso de dos modelos que utilizan arquitecturas diferentes. El que mejores resultados muestre en cuanto a precisión en la predicción se le aplicarán técnicas de data augmentation offline y online en busca de afinar la detección y disminuir el overfitting.

Al emplear este nuevo dataset hay que igualar *"num_classes"* a 2, siendo las clases que emplearemos aquí "mandarinas naranjas" y "mandarinas verdes".

- *SSD Inception V2*

En el primer experimento, se usa la arquitectura de detección SSD, empleando Inception v2 como extractor de características, y siendo *ssd_inception_v2_coco_2018_01_28* el modelo concreto escogido para este entrenamiento. El primer cambio que se debe realizar en el archivo de configuración es el de modificar la variable *"fine_tune_checkpoint"*, que igualaremos a la ruta donde se encuentre la carpeta descargada del modelo.

En el archivo de configuración de este se comprueba que se hace empleo por defecto de dos técnicas de data augmentation offline: random_horizontal_flip y ssd_random_crop. La primera técnica ya ha sido explicada con anterioridad. La segunda realiza un recorte aleatorio en la imagen.

```
data_augmentation_options {
    random_horizontal_flip {}
}
data_augmentation_options {
    ssd_random_crop {}
}
```

Este modelo en un principio cuenta con un batch_size igual 24. Dado que el equipo de desarrollo hace empleo de una CPU, el batch_size empleado será el máximo que permita el sistema. El problema de esto será que la velocidad de entrenamiento será muy reducida.

Se lleva a cabo la prueba sin modificar este parámetro, y se observa que para realizar 100 pasos el modelo tarda más de 1 hora, lo que lo convierte en inviable para el proyecto. Es por esto por lo que se decide entonces igualar el batch_size a 1.

A continuación, se procede a ejecutar el entrenamiento, el cual se lleva a cabo durante 99655 pasos, a lo largo de 49 horas. En este punto se observa que la función de pérdida de entrenamiento disminuye muy lentamente, y se mantiene en 2 para la cantidad de pasos realizada.

Como se ha comentado en la sección anterior, durante el reentrenamiento del modelo se van realizando evaluaciones a la vez que se guardan puntos de control, cada 10-15 minutos,

pintándose los resultados en formato COCO en la misma terminal donde se está ejecutando el archivo "*model_main.py".* Cuando se termina el entrenamiento, las métricas pintadas en la terminal coinciden con las de la imagen siguiente. Estos resultados, y los que se obtendrán en los siguientes experimentos, se analizarán con mayor profuncidad en el capítulo 5.1.

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.101
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.343
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.034
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.101
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.302
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.018
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.116
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.205
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.204
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.338
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
```

Imagen 29. Resultados de la evaluación en el último punto de control antes de terminar el entrenamiento. Como se denota, emplea las métricas de COCO, explicadas anteriormente.

- *Faster RCNN Inception V2*

El siguiente experimento se basará en hacer empleo de la arquitectura Faster R-CNN con el mismo extractor de características que en el caso anterior, Inception v2. El modelo preconfigurado usado es *faster_rcnn_inception_v2_coco_2018_01*_28, el cual ya se empleó antes en la prueba de Kaggle. Con este modelo preconfigurado viene por defecto un batch size igual a 1, así que no se realiza ningún cambio aquí.

Para esta prueba se debe modificar la ruta a la que se iguala el parámetro *"fine_tune_checkpoint"*, escogiendo la adecuada para este modelo.

En este experimento se realizan 25301 pasos, que se llevan a cabo a lo largo de 36 horas. Desde los 14500 pasos la función de pérdida converge a más o menos 0.2, siendo este un buen ajuste. Para este punto, las medidas de AP y AR han sufrido evidentes mejoras respecto del experimento anterior, que se basaba en la arquitectura SSD, como se analizará en el capítulo 5.1. Por este motivo, en los siguientes experimentos se decide primar la arquitectura Faster R-CNN sobre SSD.

En los dos últimos experimentos, que harán uso de Faster R-CNN Inception V2, se emplearán las dos técnicas de aumentos de datos vistas antes: offline y online.

- *Faster RCNN Inception V2 con aumento de datos offline*

En el experimento que se hace uso de offline data augmentation, los cambios realizados en el archivo de configuración son mínimos. Simplemente se aumenta el número de ejemplos de entrenamiento hasta 54, coincidiendo con las imágenes contenidas en la carpeta *"…/object_detection/images/"*, en la cual se habrán sustituido las carpetas *"images/test/"* y *"images/train/"* con las imágenes obtenidas al realizar las modificaciones manuales. También se

eliminará la técnica de aumento online que viene por defecto en el archivo de configuración, *"random_horizontal_flip"*, ya que coincide con una de las técnicas de aumento offline empleadas.

Se requiere de un total de casi 45 horas para llevar a cabo este entrenamiento, obteniendo un pérdida de 0.4 en el momento en el que se finaliza, a los 31600 pasos. Se observa una cierta mejora en la tasa de aciertos para las medidas de COCO para area de tamaño medio.

- *Faster RCNN Inception V2 con aumento de datos online*

Para el último experimento, el cual empleará online data augmentation sobre el conjunto customizado de imágenes, se volverá a hacer uso del dataset no aumentado, realizando las modificaciones pertinentes en las carpeta que contiene las imágenes y archivos .xml. En este experimento se realizarán los aumentos automáticamente, haciendo uso de configuraciones internas de la api. La variable *"num_examples"* del archivo de configuración volverá a ser 15 para así coincidir con las imágenes contenidas en la carpeta *"images/test/"* y, en este mismo archivo, se considerarán tres técnicas de aumento de datos, que concordarán con las empleadas en el aumento offline del experimento anterior. Una de ellas es la técnica de aumento que viene por defecto en la configuración del modelo empleado, *"random_horizontal_flip"*. A parte, se añaden otras dos técnicas al archivo de configuración que acompañarán a la técnica que viene por defecto: *"random_vertical_flip"* y *"random_rotation90"*.

Una vez preparado el archivo de configuración se pasa a inicializar el entrenamiento. Este se lleva a cabo durante 54800 pasos en 73 horas, obteniendo la mejor tasa tanto de aciertos como de recuperación. Converge a 0.35.

El análisis de datos obtenidos durante el entrenamiento y de los resultados, tanto de precisión teórica, con métricas de COCO y Pascal VOC, como de precisión visual, al ejecutar las inferencias sobre imágenes que no se han empleado en el entrenamiento, se tratará en el capítulo siguiente.

5. Resultados

En este capítulo se realiza un análisis de los entrenamientos de los experimentos llevados a cabo, realizando una comparativa entre las curvas de pérdida de entrenamiento y evaluación, y cotejando tanto los resultados de predicción obtenidos en las evaluaciones paralelas al entrenamiento, como los visuales al ejecutar las inferencias sobre imágenes. Por otro lado se analizará la eficiencia al ejecutar la inferencia sobre distintas unidades de procesamiento, entre ellas la CPU y GPU integradas en el equipo de desarrollo, y un acelerador USB de Intel conocido como Myriad.

5.1. Análisis de los resultados de los distintos entrenamientos

Una vez que se han completado los entrenamientos realizados con las distintas configuraciones, es importante realizar una comparativa de los valores obtenidos.

Un punto importante a tener en cuenta a la hora de elaborar este proyecto es la velocidad a la que se realiza un entrenamiento, ya que esto puede suponer un punto de inflexión a la hora de la elección del modelo. En este trabajo han habido diferencias notorias en los tiempos empleados para llegar a la convergencia de la función de pérdida de entrenamiento y, debido a un pequeño overfitting, ha llegado a disminuir levemente la tasa de aciertos a medida que avanzaba el entrenamiento, al disminuir la generalización. A continuación, se puede observar una comparativa entre los pasos realizados en una hora para cada experimento, donde se obtiene una velocidad de entrenamiento de más de 2000 pasos por hora para el modelo que hace empleo de la arquitectura SSD. Esto implica una mejora de x3 respecto a los resultados obtenidos con la arquitectura Faster R-CNN, los cuales se asemejan bastante entre sí, siendo el experimento que hace uso del aumento online el más lento debido a la modificación de las imágenes realizadas sobre la marcha.

	No aumento SSD	No aumento Faster R-CNN	Aumento Offline Faster R-CNN	Aumento Online Faster R-CNN
Pasos por hora	2033.77	750.8	750.2	702.7

A la hora de realizar los entrenamientos se ha buscado reducir el overfitting probando distintos modelos preentrenados y técnicas de aumento de datos. En la imagen 30 se pueden ver las gráficas de las funciones de pérdida de entrenamiento y evaluación de los experimentos sin aumentos para las dos arquitecturas, SSD y Faster R-CNN. Se puede comprobar que para la arquitectura SSD (arriba) ambas pérdidas son muy altas, incluso después de haber realizado casi 100 mil pasos (el

learning rate que viene por defecto en estos modelos disminuye a partir de los 90 mil pasos, por tanto, no tiene sentido continuar con la mejora). Como se ve en la función de pérdida de evaluación, el ovefitting es bastante alto al haber mucha diferencia entre las dos funciones. Este es uno de los motivos por los que los resultados de este experimento no son los deseados. Por otro lado, Faster R-CNN (abajo) sí que consigue unos resultados bastante buenos, alcanzando la pérdida de entrenamiento el valor 0.2 a los 16 mil pasos, con una pérdida de evaluación creciente, con un valor igual a 1.91 en el momento del fin del entrenamiento. Esto supone un ligero overfitting en comparación con el que ocurre con SSD.

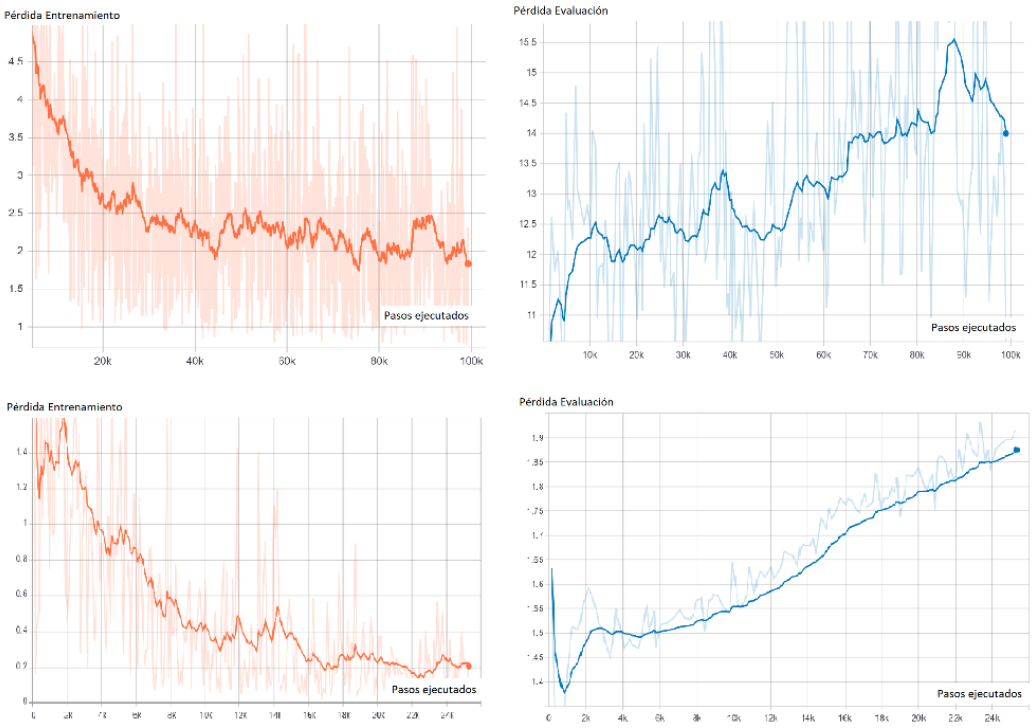

Imagen 30. Gráficas de pérdida de entrenamiento (izquierda) y evaluación (derecha) de las arquitecturas SSD Inception V2 (arriba) y Faster R-CNN V2 (abajo). La función sombreada se corresponde a los resultados reales de las pérdidas, pero como esta tiene mucho ruido, se emplea un parámetro conocido como threshold (umbral) para que la función se lea mejor, resultando la curva con un color más oscuro. Este parámetro es facilitado por Tensorboard.

Por otro lado, se encuentran las gráficas para los aumentos offline y online de Faster R-CNN que, pese a que emplean los mismos aumentos, la duración del entrenamiento es diferente para ambas como puede observarse en la imagen 31. El entrenamiento del aumento offline se finaliza pese a que la curva de entrenamiento tiende a seguir disminuyendo, a los 31.6 mil pasos. Esto se debe a que no se ha denotado ninguna mejoría en cuanto a precisión desde el paso 15 mil, incluso la tasa de detección empeora (muy levemente), implicando esto que continuar con el entrenamiento no tiene sentido. Para el aumento online ocurre algo similar, se anota que es en el paso 27790 cuando se realiza la mejor detección, obteniéndose 0.64 para la métrica de Pascal (IoU=0.50), ligeramente superior a los 6.10 finales. En ambas se denota la presencia de overfitting, pero este no es demasiado pronunciado, siendo inferior incluso al obtenido con Faster R-CNN sin aumentos.

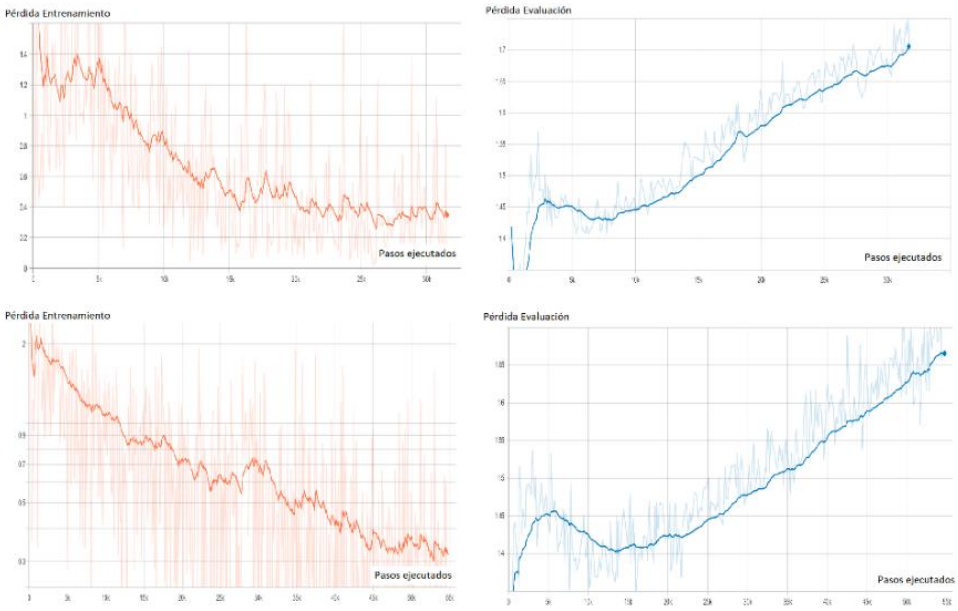

Imagen 31. Gráficas de pérdida de entrenamiento (izquierda) y evaluación (derecha) de la arquitectura Faster R-CNN Inception V2 con aumento offline (arriba) y aumento online (abajo).

Estas gráficas de las funciones de pérdida tanto de entrenamiento como de evaluación se encuentran localizadas en el repositorio GitHub[70].

El punto más importante de cara a alcanzar el objetivo de este proyecto es obtener buenos datos de precisión en el entrenamiento. Es por esto por lo que se realizan dos análisis, uno teórico, basándose en los resultados de las evaluaciones realizadas en paralelo al entrenamiento, y otro visual, ejecutando la inferencia mediante un Jupyter Notebook sobre unas imágenes que no han sido empleadas en el entrenamiento.

En el primer análisis, el teórico, se hace empleo de los resultados en formato COCO mencionados anteriormente, que son pintados en la terminal donde se ha ejecutado el entrenamiento. De los 6 parámetros de precisión media de evaluación que son mostrados, los dos que más nos interesan son la precisión media (AP) tanto para *"IoU=0.50, area=all"*, que coincide con la métrica de Pascal VOC, como para la medida de COCO *"IoU=0.50:0.95, area=medium"*. Nos interesa el primero porque, como se explica en el apartado de las métricas de evaluación, no buscamos una medida que emplee un IoU muy restrictivo ya que el reducido tamaño de las imágenes puede llevar a error en la evaluación. En cuanto a la segunda medida, nos interesa porque es la medida estándar de COCO para objetos de tamaño medio, es decir, los que se encuentran entre 32x32 y 96x96 píxeles, que es el rango en el que se encuentran la gran mayoría de los objetos contenidos en nuestras imágenes. También aparecen objetos de menor tamaño, pero para estos el IoU=0.50 es demasiado alto para cumplir con las métricas utilizadas, por lo que los resultados para estos no son contemplados. En la siguiente tabla se puede ver una comparativa de las precisiones medias obtenidas para cada arquitectura.

	No aumento SSD	No aumento Faster R-CNN	Aumento Offline Faster R-CNN	Aumento Online Faster R-CNN
Pascal VOC (IoU = 0.50)	0.343	0.542	0.540	0.610
COCO (IoU = 0.50:0.95) Area = medium	0.302	0.433	0.528	0.599

Se puede observar que los resultados para Faster R-CNN con aumento online, pese a que estos aumentos coinciden con los empleados en el aumento offline, son superiores. Esto se debe a que el aumento online realiza modificaciones para aleatorias para cada imagen, provocando que la generalización sea mejor que al usar el aumento de datos offline. Por otro lado, se denota también que Faster R-CNN con aumento offline es semejante en la medida de Pascal VOC a la misma arquitectura sin emplear aumentos, pero mejora notablemente en cuanto a la medida de COCO. Hay que recordar que la configuración por defecto de esta arquitectura por defecto tenía una técnica de aumento que al realizar la prueba con aumento offline se decidió suprimir para que los aumentos fueran solo de este último tipo.

Para finalizar el análisis, se realizará una comparativa entre los resultados visuales obtenidos al ejecutar las inferencias resultantes en cada experimento sobre tres imágenes diferentes. En la Imagen 29 se muestran en cada línea los bounding boxes generados para cada imagen, coincidiendo cada columna con un experimento.

En la primera imagen evaluada, pese a que las mandarinas no se ven muy bien a simple vista, existe una alta tasa de acierto para los distintos experimentos, aunque como era de suponer, en los experimentos que usan Faster R-CNN se consiguen más detecciones. Los resultados para la segunda imagen son muy similares.

Por otro lado, la última imagen, al tener una iluminación muy inferior que el resto, obtiene unos resultados mucho peores para la arquitectura SSD, lográndose una sola detección, cuando con las otras arquitecturas se detectan en torno a 15 mandarinas. Esto se debe a que la configuración de los modelos preentrenados SSD de Tensorflow Object Detection API hacen uso de un redimensionado 300x300 por defecto al emplear una arquitectura SSD300, reduciendo mucho la calidad de las imágenes, provocando que en esta imagen en concreto los resultados sean tan malos. Existe otra arquitectura SSD, conocida como SSD512, que puede emplearse para mejorar los resultados, pero al no encontrarse entre los modelos preentrenados, habría que realizar el entrenamiento desde cero al no poder usar la transferencia de aprendizaje.

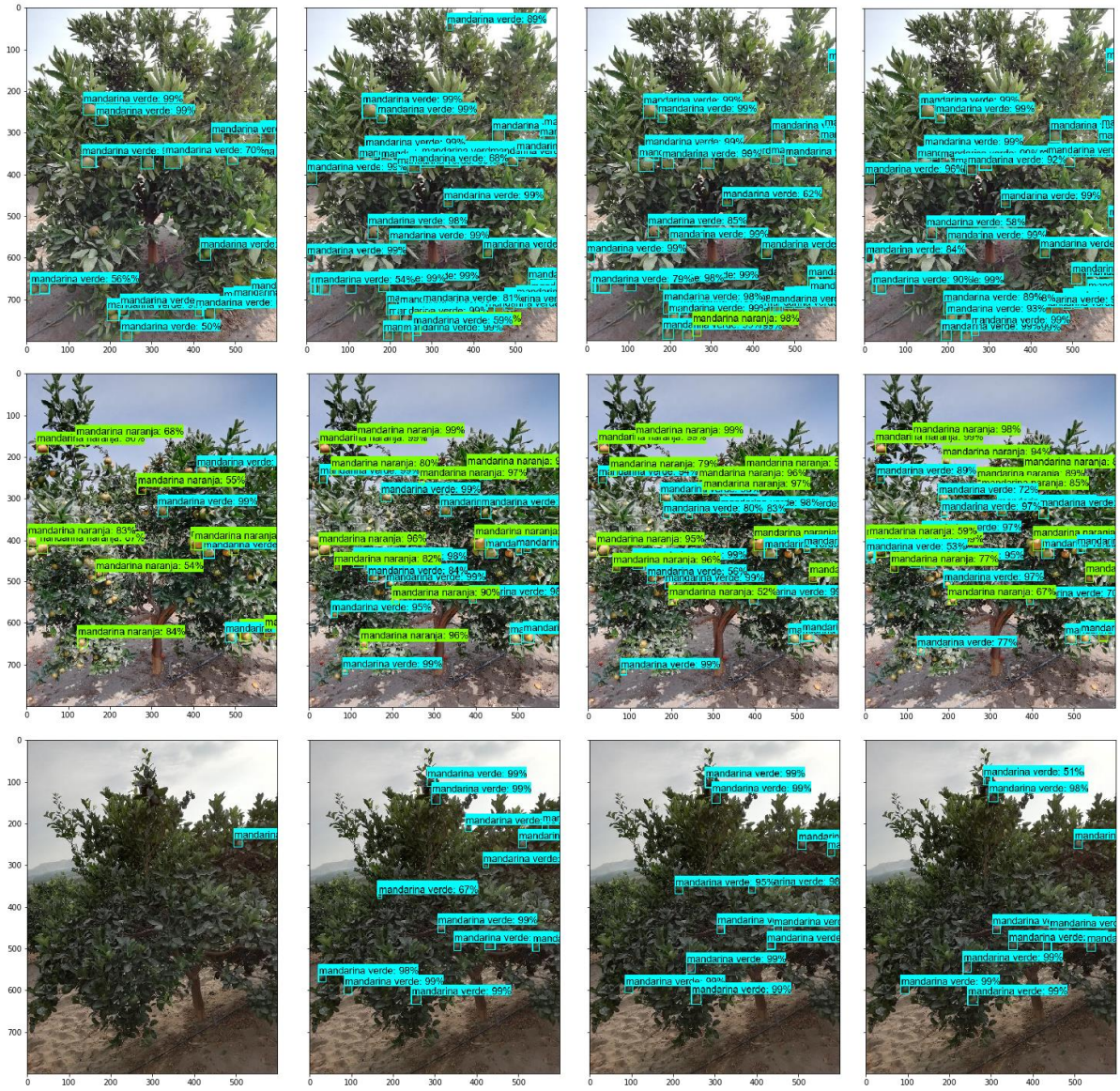

Imagen 32. Comparativa de resultados de las distintas inferencias generadas para una imagen. De izquierda a derecha, SSD, Faster R-CNN, Faster RCNN con aumento offline y Faster R-CNN con aumento online. Las mandarinas verdes se encuentran etiquetadas de color azul, mientras que las de color naranja se etiquetan de color verde.

Las imágenes originales se encuentran en *"…/object_detection/test_images"*, que es desde donde son llamadas por los Jupyter Notebook para mostrar correctamente los resultados. Tantos los cuadernos de Jupyter como la carpeta con las imágenes se encuentran localizadas en el repositorio GitHub [71].

5.2. Eficiencia de la ejecución de inferencia

Como paso complementario a lo visto en el punto anterior, se ha llevado a cabo un análisis comparativo de la velocidad de ejecución de las dos arquitecturas empleadas, SSD y Faster R-CNN, ambas con extractor de características Inception V2.

Una alta velocidad de ejecución puede implicar llevar a cabo una detección de objetos en video a tiempo real con una tasa de FPS que puede incluso llegar a ser lo suficientemente alta como para que su visualizado sea no errático.

Para realizar este análisis comparativo se hace uso del toolkit OpenVINO de Intel sobre las inferencias (archivo con extensión .pb) que vienen por defecto con los modelos preentrenados que han sido empleados en la transferencia de aprendizaje. Esto se debe gracias a que esta herramienta ofrece soporte a distintos frameworks como Caffe o Tensorflow, y por ello también a la api que se ha utilizado anteriormente. Se llevarán a cabo dos pasos en busca de realizar este análisis.

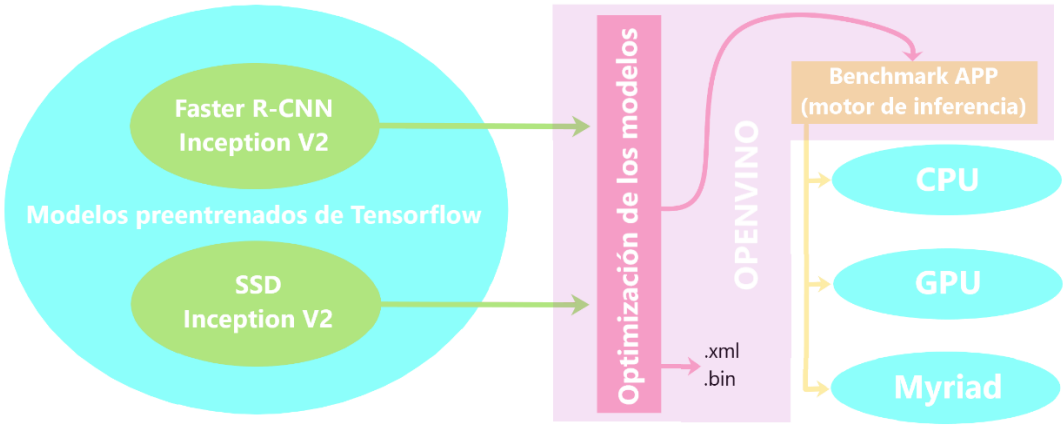

Imagen 33. Esquema del análisis realizado

El primero consistirá en convertir la inferencia en formato Tensorflow al conocido como Representación Intermedia (IR) de la red, que consistirá en tres archivos (.bin, .mapping y .xml). A esto se le conoce como optimización del modelo, el cual podrá emplearse ahora con las distintas aplicaciones que ofrece la api de OpenVINO conocida como motor de inferencia (del inglés, Inference Engine).

Para realizar esta conversión se ejecuta el comando "mo_tf.py" junto a los flags correctos, siendo de vital importancia elegir la ruta de la inferencia (.pb) y el archivo de configuración (.config), ambos encontrados en el modelo descargado, el tamaño de redimensionado de las imágenes del que hace empleo el modelo (300x300 para SSD y 600x1224 en Faster R-CNN), la ruta donde queremos que se guarden los datos de salida y el tipo de los datos (se ha escogido FP16).

Una vez se ha llevado a cabo el optimizado de los modelos, se pasan a ejecutar con la aplicación Benchmark C++ App del motor de inferencia de Openvino [72]. Esta se ejecutará con tres arquitecturas de Intel distintas, entre las que se encuentran la CPU y GPU integradas en el equipo de trabajo, Intel Core i7-7660U y Intel Iris Plus Graphics 640, respectivamente. La arquitectura restante se corresponde al USB acelerador VPU Intel® Movidius™ Myriad™ X facilitado por el profesor. Los resultados obtenidos se pueden observar a continuación.

	Intel Core i7-7660U (CPU)	Intel Iris Plus Graphics 640 (GPU)	Intel Myriad Movidius X (VPU)
SSD Inception V2	10.94 FPS	26.34 FPS	20.98 FPS
Faster R-CNN Inception V2	2.28 PFS	4.47 FPS	1.72 FPS

Como puede observarse, los mejores resultados en cuanto a velocidad de ejecución son obtenidos por los modelos que cuentan con la arquitectura SSD, siendo el mejor resultado el obtenido con la GPU integrada, con un total de 26.34 FPS, lo que implica que se podría procesar un vídeo a tiempo real sin perder calidad en su visualizado. Los resultados con el acelerador de Intel varían en función del modelo, obteniendo cerca del doble de FPS para SSD respecto de la CPU, mientras que los resultados empeoran al usar Faster R-CNN.

Los resultados obtenidos con el dispositivo Myriad son austeros frente a los obtenidos con los procesadores integrados en el equipo de desarrollo, sobretodo al compararlos con los de la GPU. Pero donde este acelerador USB destaca es en cuanto a consumo de energía, ya que haciendo una estimación partiendo del TDP de los tres procesadores, el valor para el Myriad resulta ser 10 veces menos que el resto al ser su TDP igual a 1.5 vatios, mientras que el TDP de la CPU y la GPU empleadas es de 15 vatios. Este bajo consumo puede llegar a ser un punto diferencial si, para la tarea requerida, no es necesario el uso de una alta tasa de FPS.

6. Conclusiones

Este proyecto ha supuesto una primera toma de contacto con el campo de la inteligencia artificial y, en concreto, el computer vision. Para llevarlo a cabo ha sido necesario realizar un estudio profundo de las distintas dependencias, técnicas y herramientas necesarias para el desarrollo, logrando así disponer de una visión global de esta rama de la informática.

La idea en la que se basa el proyecto, el diseño de un programa que logre detectar frutos en árboles, se encuentra englobada en la técnica de visión por computador conocida como detección de objetos, para la cual se dispone de una alta gama de apis y frameworks específicos. El pequeño tamaño en píxeles de los objetos a detectar y su mimetización con el fondo de la imagen, sumados a la reducida cantidad de imágenes que conforman el dataset custom, han derivado en la necesidad de analizar distintos modelos preentrenados con distintas arquitecturas con el fin de obtener los mejores resultados en cuanto a precisión, apoyándose en la transferencia de aprendizaje. Los resultados obtenidos con la arquitectura Faster R-CNN superan con creces los de SSD, alcanzando una precisión media de 0.6 sobre 1.0, lo que supone el doble que la precisión de la segunda.

Además se ha realizado un acercamiento al hardware específico para la aceleración en ejecución empleando la herramienta OpenVINO en busca de hacer una comparativa entre las unidades de procesamiento integradas en el equipo de desarrollo y el acelerador USB conocido como Myriad, todos de Intel, observando que este último dispone del mejor ratio entre la tasa de FPS y el consumo energético. Al contrario que en los resultados de precisión, en esta comparativa se observa que la arquitectura SSD se ejecuta entorno a 5 veces más rápido con la CPU y la GPU que Faster R-CNN, diferencia que se incrementa hasta 12 al ejecutarlas con el Myriad.

Estos preliminares pero prometedores resultados abren la puerta a considerar seguir explorando la mejora del algoritmo desarrollado considerando las siguiente líneas de actuación:

- Obtener nuevas imágenes para aumentar así el tamaño del conjunto de datos de entrenamiento, disminuyendo en la medida de lo posible el overfitting y mejorando los resultados. Y, a su vez, añadir a este dataset nuevas clases, como uvas o peras, con el objetivo de aumentar las versatilidad de la herramienta.
- Llevar a cabo una mayor cantidad de experimentos con el fin de probar la precisión obtenida con otras arquitecturas y extractores de características, valiéndose de equipos de desarrollo más potentes que hagan empleo de GPU para agilizar los entrenamientos.
- Realizar un despliegue del proyecto en producción para poder llevar a cabo tareas de control de cultivos, por ejemplo, haciendo empleo de cámaras de videovigilancia, en punto fijo o ancladas en drones para, apoyándose en el dispositivo acelerador de inferencia visto en el proyecto, Myriad X, poder ejecutar el programa de detección con bajos consumos energéticos y tiempos de inferencia bastante bajos.

7. Conclusions

This project has been a first contact with the field of artificial intelligence and, specifically, computer vision. To carry out an implementation it has been necessary to make an in-depth study of the different dependencies, techniques and tools necessary for development, achieving a global vision of this branch of computer science.

The idea on which this project is based, the design of a program that manages to detect fruits on trees, is encompassed in the computer vision technique known as object detection, for which exists a high range of apis and specific frameworks. The small size in pixels in the objects to be detected and their mimicry with the background of the image, added to the small amount of images that make up the customized data set, have resulted in the need for analysis of different pre-trained models with different architectures in order to obtain the best results in terms of accuracy, based on the transfer learning. The results obtained with the Faster R-CNN architecture far exceed those of SSD, reaching an average accuracy of 0.6 over 1.0, which is twice the precision of the second.

In addition, it has been made an approach to the specific hardware for acceleration in execution using the OpenVINO tool in order to make a comparison between the processing units integrated in the development equipment and the USB accelerator known as Myriad X, observing that the latter has the best relationship between the FPS rate and energy consumption. In contrast to the precision results, this comparison shows that the SSD architecture runs around 5 times faster with the CPU and GPU than Faster R-CNN, a difference that increases up to 12 when it is executed with the Myriad.

These preliminary but promising results open the door to consider further exploring the improvement of the algorithm developed considering the following lines of action:

- Obtain new images to increase the size of the training data set, in order to decrease over-adjustment as much as possible and improve the results. And, also, add this dataset to new classes, such as grapes or pears, in order to increase the versatility of the tool.
- Carry out a greater number of experiments in order to test the accuracy obtained with other architectures and feature extractors, using more powerful development teams that will use GPUs to speed up training.
- To make a deployment of the project in production to be able to carry out a crop control tasks, for example, to use video surveillance cameras, at a fixed point or anchored in drones to run the detection program, using the inference accelerator device seen in the project, Myriad X, in order to get low energy consumption and quite low inference times.

8. Bibliografía

[1] Foro de Expertos de Alto Nivel. (2009). La agricultura mundial en la perspectiva del año 2050. FAO. http://www.fao.org/fileadmin/templates/wsfs/docs/Issues_papers/Issues_papers_SP/La_agricultura_mundial.pdf

[2] Ferrandez, Javier & Alcañiz-Lucas, Sara & García-Chamizo, Juan & Platero-Horcajadas, Manuel. (2019). Smart Environments Design on Industrial Automated Greenhouses. Proceedings. https://www.mdpi.com/2504-3900/31/1/36/pdf

[3] Miss.Snehal S.Dahikar, Dr.Sandeep V.Rode. (2014). Agricultural Crop Yield Prediction Using Artificial Neural Network Approach. https://pdfs.semanticscholar.org/7c68/a32212c1f86f535f4c1658ff68399d0a9ddd.pdf

[4] A. K. Saha et al. (2018). IOT-based drone for improvement of crop quality in agricultural field. IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, pp. 612-615. https://ieeexplore.ieee.org/abstract/document/8301662/authors#authors

[5] Kasey Panetta. (2019). 5 Trends Appear on the Gartner Hype Cycle for Emerging Technologies. https://www.gartner.com/smarterwithgartner/5-trends-appear-on-the-gartner-hype-cycle-for-emerging-technologies-2019/

[6] A. M. Turing. (1950). Computing Machinery and Intelligence. Mind 49: 433-460. https://www.turing.org.uk/scrapbook/test.html

[7] David A. Forsyth and Jean Ponce. (2002). Computer Vision: A Modern Approach. Prentice Hall Professional Technical Reference. ISBN:978-0-13-085198-7

[8] Alex Smola and S.V.N. Vishwanathan. (2008). Introduction to Machine Learning. Cambridge Univertity Press. https://alex.smola.org/drafts/thebook.pdf

[9] Sonali, Maind, B., & Wankar, P. (2014). Research Paper on Basic of Artificial Neural Network. https://www.semanticscholar.org/paper/Research-Paper-on-Basic-of-Artificial-Neural-Sonali-Maind/cb22b35b740a79ce710f7471c0bc01e570092480

[10] Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., Liu, C. (2018). A Survey on Deep Transfer Learning. https://arxiv.org/pdf/1808.01974.pdf)

[11] LeCun, Y., Bengio, Y. & Hinton, G. (2015). Deep learning. Nature 521, 436–444. https://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf

[12] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65(6), 386–408. https://psycnet.apa.org/doiLanding?doi=10.1037%2Fh0042519

[13] Ivakhnenko, A.G., Lapa, V.G., & Mcdonough, R.N. (1967). Cybernetics and forecasting techniques. https://www.semanticscholar.org/paper/Cybernetics-and-forecasting-techniques-Ivakhnenko-Lapa/15dbfe20aa7e2c3d0a48938a63afc9b810ffe4c1

[14] Ronneberger, O., Fischer, P., Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. https://arxiv.org/pdf/1505.04597.pdf

[15] Yue, T., Wang, H. (2018). Deep Learning for Genomics: A Concise Overview. https://arxiv.org/pdf/1802.00810.pdf

[16] The Theano Development Team, and 112 colleagues. (2016). Theano: A Python framework for fast computation of mathematical expressions. https://arxiv.org/pdf/1605.02688.pdf

[17] Tingwu Wang, Machine Learning Group. Semantic Segmentation. University of Toronto. http://www.cs.toronto.edu/~tingwuwang/semantic_segmentation.pdf

[18] O'Shea, Keiron & Nash, Ryan. (2015). An Introduction to Convolutional Neural Networks. https://www.researchgate.net/publication/285164623_An_Introduction_to_Convolutional_Neural_Networks

[19] Fukushima, K. (1980). Biol. Cybernetics. 36: 193. https://www.rctn.org/bruno/public/papers/Fukushima1980.pdf

[20] Alex Krizhevsky and Sutskever, Ilya and Hinton, Geoffrey E. (2012). Advances in Neural Information Processing Systems, pp. 1097-1105. https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[21] Simonyan, K., Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. https://arxiv.org/pdf/1409.1556.pdf

[22] He, K., Zhang, X., Ren, S., Sun, J. (2015). Deep Residual Learning for Image Recognition. https://arxiv.org/pdf/1512.03385.pdf

[23] Szegedy, C., and 8 colleagues. (2014). Going Deeper with Convolutions. (https://arxiv.org/pdf/1409.4842.pdf)

[24] ImageNet project. http://image-net.org/about-overview

[25] Uijlings, J.R.R., van de Sande, K.E.A., Gevers, T. et al. (2013). Selective Search for Object Recognition. Int J Comput Vis 104, 154–171. http://www.huppelen.nl/publications/selectiveSearchDraft.pdf

[26] R. Girshick, J. Donahue, T. Darrell and J. Malik. (2016). Region-Based Convolutional Networks for Accurate Object Detection and Segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 38, no. 1, pp. 142-158. (http://islab.ulsan.ac.kr/files/announcement/513/rcnn_pami.pdf)

[27] He, K., Zhang, X., Ren, S., Sun, J. (2014). Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. https://arxiv.org/pdf/1406.4729.pdf

[28] Girshick, R. (2015). Fast R-CNN. (https://arxiv.org/pdf/1504.08083.pdf)

[29] Ren, S., He, K., Girshick, R., Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. https://arxiv.org/pdf/1506.01497.pdf

[30] Redmon, J., Divvala, S., Girshick, R., Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. https://arxiv.org/pdf/1506.02640.pdf

[31] Redmon, J., Farhadi, A. (2016). YOLO9000: Better, Faster, Stronger. https://arxiv.org/pdf/1612.08242.pdf

[32] Redmon, J., Farhadi, A. (2018). YOLOv3: An Incremental Improvement. https://arxiv.org/pdf/1804.02767.pdf

[33] Liu, W., and 6 colleagues. (2015). SSD: Single Shot MultiBox Detector. https://arxiv.org/pdf/1512.02325.pdf

[34] Chen, L.-C., Hermans, A., Papandreou, G., Schroff, F., Wang, P., Adam, H. (2017). MaskLab: Instance Segmentation by Refining Object Detection with Semantic and Direction Features. https://arxiv.org/pdf/1712.04837.pdf

[35] Torch. https://github.com/torch/torch7

[36] PyTorch. https://github.com/pytorch/pytorch

[37] Tensorflow. https://www.tensorflow.org/

[38] Caffe. https://github.com/BVLC/caffe

[39] Caffe2. https://caffe2.ai/docs/caffe-migration.html

[40] OpenVINO toolkit. https://docs.openvinotoolkit.org/latest/_docs_install_guides_installing_openvino_windows.html

[41] Schlegel, Daniel. (2015). Deep Machine Learning on GPUs. University of Heidelberg. http://www.ziti.uni-heidelberg.de/ziti/uploads/ce_group/seminar/2014-Daniel_Schlegel.pdf

[42] NVIDIA cuDNN. https://developer.nvidia.com/cudnn

[43] Intel MKL. https://software.intel.com/en-us/mkl

[44] Google Cloud AI Platform. https://cloud.google.com/ai-platform/?hl=es-419

[45] Microdoft Azure Machine Learning. https://azure.microsoft.com/es-es/services/machine-learning/

[46] Machine Learning en AWS. https://aws.amazon.com/es/machine-learning/

[47] Li, E., Zeng, L., Zhou, Z., Chen, X. (2019). Edge AI: On-Demand Accelerating Deep Neural Network Inference via Edge Computing. https://arxiv.org/abs/1910.05316

[48] Gaurav Batra, Zach Jacobson, Siddarth Madhav, Andrea Queirolo, and Nick Santhanam. (2019). Artificial-intelligence hardware: New opportunities for semiconductor companies. https://www.mckinsey.com/~/media/McKinsey/Industries/Semiconductors/Our%20Insights/Artificial%20intelligence%20hardware%20New%20opportunities%20for%20semiconductor%20companies/Artificial-intelligence-hardware.ashx

[49] Edge TPU de Google Cloud. https://cloud.google.com/edge-tpu/?hl=es

[50] USB Accelerator de Coral, Google. https://coral.ai/products/accelerator/

[51] Intel® Neural Compute Stick 2. (Intel® NCS2) https://software.intel.com/es-es/neural-compute-stick

[52] Pascal Visual Object Classes project. http://host.robots.ox.ac.uk/pascal/VOC/

[53] ImageNet project. http://image-net.org/about-overview

[53] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba. (2017). Places: A 10 million Image Database for Scene Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence. http://places2.csail.mit.edu/

[54] Open Images Dataset, Google AI. https://ai.googleblog.com/2016/09/introducing-open-images-dataset.html

[55] Visual Genome project. http://visualgenome.org/static/paper/Visual_Genome.pdf

[56] Lin, T.-Y., and 9 colleagues. (2014). Microsoft COCO: Common Objects in Context. https://arxiv.org/pdf/1405.0312.pdf

[57] Lionbridge, Image Annotation Services. https://lionbridge.ai/services/image-annotation/

[58] Herramienta de anotación de imágenes LabelImg. https://github.com/tzutalin/labelImg

[59] Oksuz, K., Cam, B.~C., Akbas, E., Kalkan, S. (2018). Localization Recall Precision (LRP): A New Performance Metric for Object Detection. https://arxiv.org/pdf/1807.01696.pdf

[60] Métricas de evaluación de COCO. http://cocodataset.org/#detection-eval

[61] Kaggle Fruit Images Dataset for Object Detection. https://www.kaggle.com/mbkinaci/fruit-images-for-object-detection

[62] Dataset custom https://github.com/alotorres/Proyecto/tree/master/6-Dataset

[63] Connor Shorten, Taghi M. Khoshgoftaar. (2019). A survey on Image Data Augmentation for Deep Learning. https://link.springer.com/content/pdf/10.1186%2Fs40537-019-0197-0.pdf

[64] Scripts para el aumento del conjunto de imágenes. https://medium.com/@bhuwanbhattarai/image-data-augmentation-and-parsing-into-an-xml-file-in-pascal-voc-format-for-object-detection-4cca3d24b33b

[65] Tensorflow Object Detection API. https://github.com/tensorflow/models/tree/master/research/object_detection

[66] Repositorio de Tensorflow Models. https://github.com/tensorflow/models

[67] Repositorio del archivo xml_to_csv.py
https://gist.github.com/TannerGilbert/9675971bcb1e8f64dbc8367473ff51ec#file-xml_to_csv_changes-py

[68] TFRecord de Tensorflow. https://www.tensorflow.org/tutorials/load_data/tfrecord

[69] Tensorboard de Tensorflow. https://www.tensorflow.org/tensorboard

[70] Enlace a repositorio con las gráficas de pérdida de entrenamiento y evaluación.
https://github.com/alotorres/Proyecto/tree/master/3-Gr%C3%A1ficas%20de%20funci%C3%B3n%20de%20p%C3%A9rdida

[71] Enlace a repositorio con resultados visuales al ejecutar las distintas inferencias.
https://github.com/alotorres/Proyecto/tree/master/4-Pruebas

[72] Aplicación Benchmark de OpenVINO.
https://docs.openvinotoolkit.org/latest/_inference_engine_samples_benchmark_app_README.html